SOFTBANK MOOK

1999春号

Oh!X

特集：ネットワークゲームの地平へ

その可能性と問題点を分析する／負荷とゲームデザイン
DirectPlay/Javaサーバアプリ/CGIによるゲームの実装

もう一度CPUについて考えてみよう／CGAコンテスト結果発表
特別企画：PlayStation2はビジュアルを革命するか
特別付録：CD-ROM 2 枚組

SOFT BANK Publishing
オー！エックス
1999春号
定価2800円

SOFTBANK MOOK

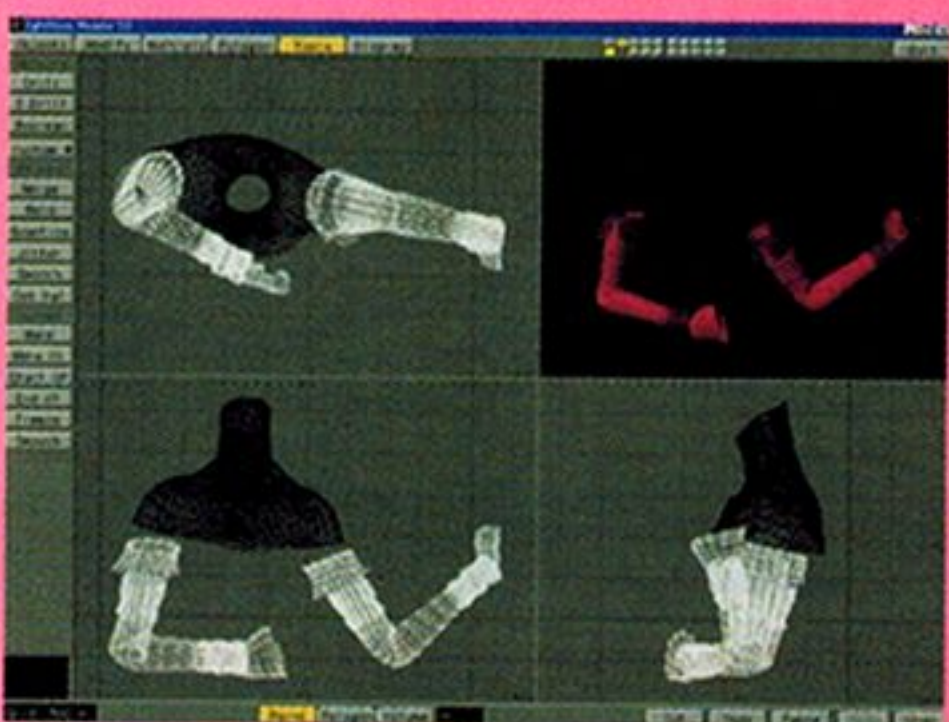
1999 春号

Oh!X

特集　ExpertMission

特集　Quake2 Bot

テレビアニメの作り方

Tcl/Tk ミニミニゲーム

VisualLaboratory

バンプマッピング

C O N T

4　序

●特別企画　PlayStation2 の予備研究

6　PlayStation2 はビジュアルを革命するか　中野修一

●特集　ネットワークゲームの地平へ

176　認識の境界を超えて　中野修一

180　多体問題と相互作用を考える　三沢和彦

186　Diablo に見るネットワークゲーム構成法　瀧　康史

192　インターネットセキュリティ事件簿　三沢和彦

194　インターネットゲーム時代のコミュニケーション論　古村　聡

198　海外コインオペゲームにおける通信　市川幹人

200　アーケードゲームに見る通信対戦の未来　如月　緑

205　DirectPlay を使った対戦ゲーム　菊地　功

210　Java ネットゲー考察 ExpertMission 制作編　霧　雨

216　ExpertMission ユーザーズマニュアル　野沢絵美

220　CGI による継続したユーザー管理　大和　哲

224　Macintosh をサーバにしてゲームを作ろう　古籏一浩

228　ネットワーク対戦エミュレータ“Bot”　須藤芳政

232　第 2 の人生，Ultima Online　kazuhisa@Pacific

●特別企画　もう一度 CPU について考えてみよう

262　CISC と RISC，そしてメモリの物語　中野修一

266　V シリーズに見る CISC の幻想と伝説　中森　章

274　VR シリーズの伝説 MIPS RISC の系譜　中森　章

● Oh!X68000

164　X680x0 再入門　あいはらてつや

168　これからの X680x0，そして零式　中村隆生

173　零式 OS 構想　鎌田　誠

● Step to the Black Arts

Level 1

80　VB でテキストエディタを作ってみる　中野修一

84　Future BASIC による Macintosh プログラム制作　第 2 回　画像の表示とアニメーション　古籏一浩

97　Tcl/Tk お気楽 GUI プログラミング入門編　広井　誠

115　Tcl/Tk によるミニミニゲーム集　広井　誠

118　VisualC++ で始める Windows プログラミング　第 2 回　MDI の基礎と画像表示　菊地　功

132　C++Builder で書く Windows プログラム　岡田智宣

138　FLASH でゲームを作ろう！　伊藤のりゆき

146　JavaScript から始めるプログラミング　第 2 回　Macromedia FLASH の制御　古籏一浩

Level 2

238　Direct3D Retained Mode 講座　第 2 回　2 段レンダリング vs インタポレータ　菊地　功

246　DirectMusic を使う　永島ひとし

E N T S

252	x86アセンブラ講座 第2回 MMX命令を使ってみよう	影山裕昭
Level 3		
284	仏日翻訳プログラム 来来仏語その2	石上達也
290	D3D Immediate Modeを使う 第1回 いきなりバンプマッピング	菊地 功
298	言語処理プログラムを作る 第1回	石上達也
●Visual Laboratory		
18	お魚を描く	川原由唯
24	天使遊戯	都築和彦
31	夢のハワイ	末次徹朗
34	みのり	森川久志
45	My Friend	田中順子
50	映画のように	由水 桂
152	テレビアニメ(CGパート)の作り方	かまたゆたか
158	QuickTimeVRの理論と実践	荻窪 圭
●Gamer's Eye		
58	Flipper Pinball Stories #2 Plandgerの変遷を見る	市川幹人
64	マイクロソフトピンボールアーケード	市川幹人
70	ビストロ2に見る食べ物ゲームの至福	西尾ゆき
74	究極の箱庭か? ポピュラス・ザ・ビギニング	西川善司
●連載/その他		
282	御託僕鐸annex 本音と建前の間で	小笠原陽介
306	拡張電脳学入門講座 清く正しく21世紀	祝 一平
308	知能機械概論 第101回 Papert博士からの2つの贈りもの StarLOGOとMINDSTORM	有田隆也
54	第11回アマチュアCGAコンテスト入選作品発表	
77	The USER'S WORKS Duel Fighter 2	
259	付録CD-ROMの使い方	
305	Oh!X 1999春号読者モニター/プレゼント	
314	STUDIO X	
318	編集室から	

＜スタッフ＞
●編集/植木章夫 ●編集協力/水上智雄 ●協力/有田隆也 石上達也 市川幹人 伊藤のりゆき 祝 一平 岡田和久 岡田智宣 小笠原陽介 影山裕昭 如月 緑 霧雨 古村 聡 末次徹朗 須藤芳政 瀧 康史 田中順子 都築和彦 中森 章 永島ひとし 西尾ゆき 野沢絵美 広井 誠 福原 徹 古旗一浩 御木徳高 三沢和彦 森川久志 吉田賢司 吉田幸一 由水 桂 プロジェクトチームDoGA 満開製作所
●カメラ/新井邦彦 ●校正/フィールドアップ
●イラスト/岡村直也 高橋哲史 塚田哲也 福原 徹 山田晴久
●編集長/来島 樹 ●編集人/高橋憲一郎 ●発行人/岡崎 眞
●DTPデザイン/クニメディア ●印刷/クニメディア

FACE TANK

フェイスタンクは大気圏中に蔓延する人間の出した"邪気"を吸い取る目的で開発された戦車型の機械である。通常は人気のない場所で静かに吸引作業を行っているが,"邪気"が溜まってくると青色の頭部が次第に赤色を帯びてくるのが特徴である。

また"邪気"の多い人間が近くにいる場合,すぐさま吸引用のダクトを伸ばしてその人間の頭部に直接あてがい,吸引作業を行うこともある。

フェイスタンクの走行速度は意外と速く,最大200km/時をマークする。すれちがいざまに吸引することも多いので,吸引されても気がつかない場合も多い。

表紙 塚田哲也
JCGL出身のCGデザイナー。

代表作

●スチール(雑誌/ポスター)
「Oh!X」「ニューズウィーク日本版」
「03(ゼロサン)」「Windows World」
「PLAYBOY日本版」「Quark」「Vi Vi」
「Tarzan」「バズラー」「ログイン」
「ポプコム」「PCマガジン」 他
「カルチャーコロンブス」(渋谷西武ポスター)

●アニメーション
「おもいッきりテレビ」(日本テレビ、オープニングタイトル)
「資生堂 枝毛用シャンプー(キューティクル効果編)」(CM)
「ベスト、モータリング」(ビデオ用オープニングタイトル)
「O2アイランド」(テーマパーク用VTR) 他
最近、CGビデオ「デジタルブルー」を発表。

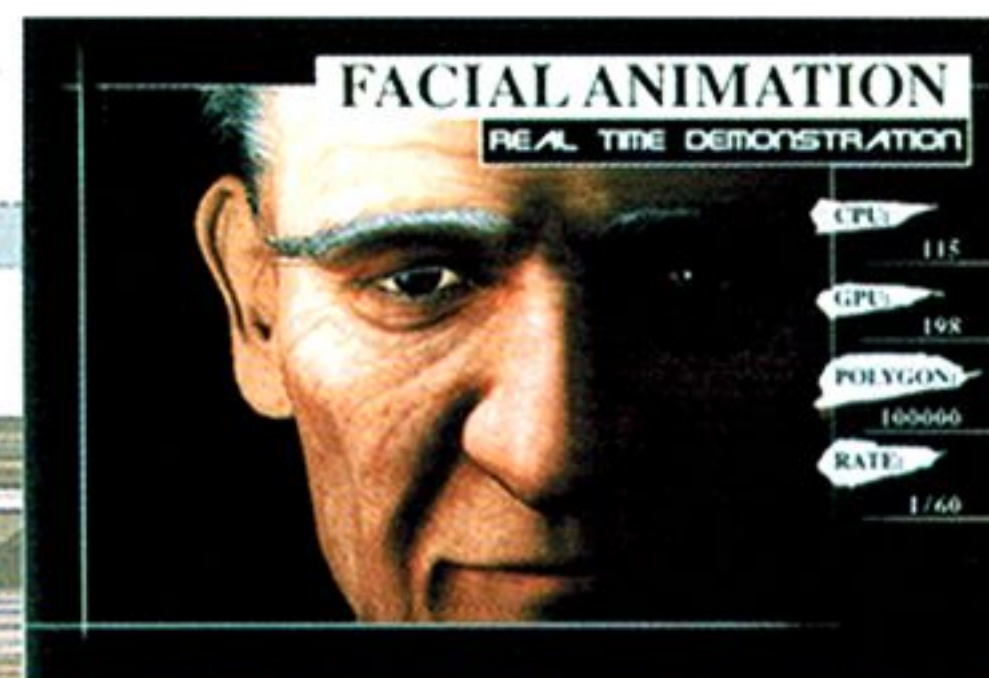
http://home.interlink.or.jp/~hotmenu

そして伝説は続く…

おかげさまでOh!X復刊号は非常に大きな反響をいただいた。お待たせしていた方々にはそれなりに楽しんでもらえたことと思う。あの時点でできることはすべて詰め込んでみたつもりだが，それでもまだまだ足りないものも多い。それは追い追いやっていくことにしよう。
それ以外に，
「こういうのを待っていた！」
という声は結構意外なところからも聞かれた。しかし，この人たちはいったいなにを待っていたのだろうか？　こういった人が「待っていた」ものと，新たに提示されたOh!Xが同じものなのだろうか？
単に，なんらかのムーブメントに飢えていただけ人もいるだろうし，まったく誤解している人もいるかもしれない。
もちろん，Oh!XがOh!Xという名前で出てきたという事実だけで十分に通じている人もいるだろう。そういう人についてはまったく問題がない。かつてのOh!Xをあまり知らない人にとっては，たぶん理解できない部分も多かったはずだ。これは知識の問題ではなく，思想の問題だからだ。
さらに，作っている側からすれば，まだ新しいOh!Xの姿をちゃんと提示しているわけではない。２号目からもっと走りたかったのだが，準備が不十分なのでまだまだもたついている。その不十分な状態で，ちゃんと理解してもらえたのかどうかはかなり不安が残っている。
それでも，なにか新しい動きに期待を持つことは悪いことではない。それだけ現状のパソコン業界は退屈だということかもしれない。
Oh!Xはなにも現在のパソコン業界であるとかゲーム業界とかを批判しようとか，対抗していこうとかいうものではない。ただ単に，我々が目指しているものと，そのほかの者が目指している場所が明確に違っているというだけの話だ。彼らは彼らの道を行けばいいし，我々は自分で道を拓いていく。多少誤解されているようだが，好んでサブカル路線を選んでいるわけでもない。むしろ我々の路線はずっと一貫している。
それでもいまの世の中ではきっと「変なことをやっている一団」にすぎないのだろう。異端とされることは別に恐れるようなことではない。ハード工作とはコネクタをつないでネジを締めることではない。プログラミングにアセンブラを使うことは必要な場合もある。CPU性能だけでマシンを語るべきではない。理論上可能なことの大半は実現が可能である。そう，常識そのものがひとつのものではないのだ。
コンピュータがこれだけ普及して生活に欠かせないものとなろうとしているにも関わらず，人間のコンピュータを扱う能力はむしろ低下してきていないだろうか。おじいちゃんおばあちゃんにも使えるパソコンやOSを作る，というのもひとつの道だろう。が，どんなパソコンやOSも使いこなせるおじいちゃんおばちゃんを作るのも，もうひとつの道だ。両者が等価だとはいわないが，どっちがより高いパフォーマンスを挙げられるかは考えるまでもないだろう。
人間とコンピュータの関係にはいろいろなかたちがあってしかるべきだ。無論，前述のような関係を世間一般に広めるべきだとはいわないが，そのような関係もありうることは明白だ。
かといって別に「クリエーターになりましょうね」というのが我々のメッセージではない。どちらかというと，「クリエーター」と呼ばれるようになったらもうダメだ。クリエイティブであることが当たり前でなくなったら人間おしまいだろう。
ところで，おめでとうという素朴なメッセージのほか，とにかく多かったのが「久しぶりにX68000を立ち上げました」という声だ。メイン環境をよそに移し替えている人が，わざわざ，しまってあったX68000を掘り返している。それでは機種を超えて展開しようとする今回のOh!Xの趣旨とはちょっと外れてしまう。
しかし，パソコンでなにかをやりたいと思い立ったときに，あい変わらず引っ張り出されるマシンなのだなあと妙に納得したりもするのだ。

特別企画

PlayStation2の予備研究

FACIAL ANIMATION
REAL TIME DEMONSTRATION

PlayStation2(仮名)の概要が公表された。
コア部分だけ見てもゲーム機としては未曾有のスケールで性能はグラフィックワークステーション並み,
カタログスペックレベルでは「RealityEngineってなんだったんだ?」と感じさせるくらいの製品である。
メモリ量などの周辺環境が追いついていない感じはあるのだが,そのポテンシャルはきわめて高い。
また,外部拡張や他製品との連携などが示唆されており,
今後の家電業界を変える製品になるかもしれないということでも注目されている。
経験的にいえば「たぶんそういうことにはならない」のだが,
一応やるつもりみたいなので(ちゃんとした構想があるのかないのかもわからないが),
ゲームに興味のない人にも関わってくるかもしれない製品である。
Oh!X読者辺りだと,ユーザーとしてはもちろん,なかには飯の種にする人も出てくるのだろう。
現段階では評価できない部分も多いのだが,あえて,SCEの提示した次世代機の姿を検証してみたいと思う。
個人的にSCEの製品はゲーム機というよりはリアルタイム画像生成機と考えたほうが理解しやすいと思っている。
多くの機能を備えるものの,中心となるのはやはり画像処理機能である。
以下では,そのスペックの示すところのものはなにを可能にするのか,
ビジュアル表現を中心にSCEの描く次世代ゲームの方向性について見ていきたいと思う。

PlayStation2は ビジュアルを革命するか

中野修一 Nakano Shuichi

さてさて，こちらの予想よりちょっと早めにPlayStation2(仮称)の発表が行われてしまった。仕様発表後しばらくしてかわされたT氏(元Oh!X/現The・PlayStation編集長)とU氏の会話。
T：「どうですか？」
U：「可愛いげがないですね」

正直な感想でいうと「よくこんなもん作ったもんだ」という感じ。作ろうと思い立ったのも凄いが，設計現場の人はさぞや苦労したことだろう。PlayStationのときもCPUコアにGTEとMDECが入っていたので，基本構成そのままの正常進化形といえばそうなんだけど……。従来のPlay Stationに対して，CPU性能と頂点演算性能は20倍，描画性能は200倍程度に強化されている。

PlayStation2の発表は，実は私を含めてOh!Xのスタッフはそれほど熱心にチェックしていなかった。Oh!Xの作業が忙しくて，発表があることは事前にわかっていても誰も動けない。
「だいたい予想つくからいいや」
という感じだった。ニュースなども見ていないし，ゲーム誌とかに載ってるのを見て，
「ふ～ん」
とかいってるようでは，業界関係者としては失格かもしれない。

しかし，意外に(当然か？)ゲーム誌でも詳しめの紹介が載っていたので，Oh!Xでいまさら取り上げる必要もないのではないか？　との意見もあったのだが，ドリキャスをやっといてPS2をやらないんでは片手落ちだろうということで，あちこちから情報をかき集めてきて，その可能性を推測してみたい。いや，当初の目論見では，発表前にOh!X2号目が出るはずだったんだけど……。

ゲーム誌の記事と同じことをやってもしかたないので，ここではもう少しいい加減な話を盛り込んでやってみたい。ゲーム誌の記事を見てもたいていどこか間違ってるし，このOh!Xに書いてある情報もたぶん結構間違ってるだろうと思われる。ま，現物が出てくるまではわからないものなので，噂話の集大成くらいのつもりで読むのがいいだろう。

Oh!XのスタッフにSCE関係者がいるのはご存じの方も多いと思うが，まさかそっちから突っ込んだ話は聞けないので，一部，変な期待をしている人には悪いのだが，情報の確度はまったく保証できない。メーカー記事チェックとかも通さないんで，写真もありあわせ。ずいぶん地味な話になるけど，ま，半分フィクションだと思うくらいでちょうどよいかも。

PlayStation2のスペック

改めてスペックを挙げてみよう。表1だ。
図1が推定されるブロックダイアグラムである。

メインバスは128ビット/150MHz，サブバスは32ビット/37.5MHzと推定される。サブバスインタフェイスを介してデータのやり取りが行われる。

メイン部はCPUチップとGPUチップ，いわゆるEmotionEngineとGraphicsSynthesizerに大別される。それに加えてI/O部分でPlayStation2は構成されている。それぞれの部分ごとに見ていこう。

●EmotionEngine

CPUコアとなる「128ビットRISC」と発表されている部分がまず謎だ。MIPS RISC系64ビットの2ウェイスーパースカラ構成で128ビットとか，ベクトルプロセッサ部分のことをいっているのかと思ったが，CPUコアの内部ブロック図を見る

表1 次世代PlayStation概要仕様書

CPU	128ビット"Emotion Engine"
クロック周波数	300MHz
キャッシュメモリ	命令：16KB，データ：8KB+16KB(SP)
メインメモリ	Direct RDRAM
メモリ容量	32MB
メモリバスバンド幅	3.2GB/秒
コプロセッサ	FPU(浮動小数点賣積和器×1，浮動小数点除算器×1)
ベクトル演算ユニット	VU0+VU1(浮動小数点積和器×9，浮動小数点除算器×3)
浮動小数点演算性能	6.2GFLOPS/秒
3次元CG座標演算性能	6600万ポリゴン/秒
圧縮画像デコーダ	MPEG2
グラフィック	"GraphicsSynthesizer"
クロック周波数	150MHz
DRAMバスバンド幅	48GB/秒
DRAMバス幅	2560ビット
ピクセル構成	RGB：Alpha：Z(24：8：32)
最大描画性能	7500万ポリゴン/秒
サウンド	SPU2+CPU
同時発音数	ADPCM：48ch(SPU2)+ソフト音源数
サンプリング周波数	44.1/48kHz
IOP	I/O Processor
CPUコア	PlayStation CPU
クロック周波数	33.8/37.5MHz
サブバス	32ビット
入出力	IEEE1394，USB
通信ポート	PCカードで対応
メディア	CD-ROM/DVD-ROM

メインDMA / IPU / メインメモリ / メインバス(128ビット) / FPU / メインCPU / VPE0 / VPE1 / Graphics Synthesizer / サブバスI/F / Emotion Engine / GPU I/F / GPU / CRTC / フレームメモリ / ビデオ出力 / サブDMA / サブCPU / サブRAM / サブROM / I/Oプロセッサ / サブバス(32ビット) / DVD-ROM / ATM / SPU2(サウンド) / PCカード / IEEE 1394 / USB / CPU / 各種I/O / メモリ

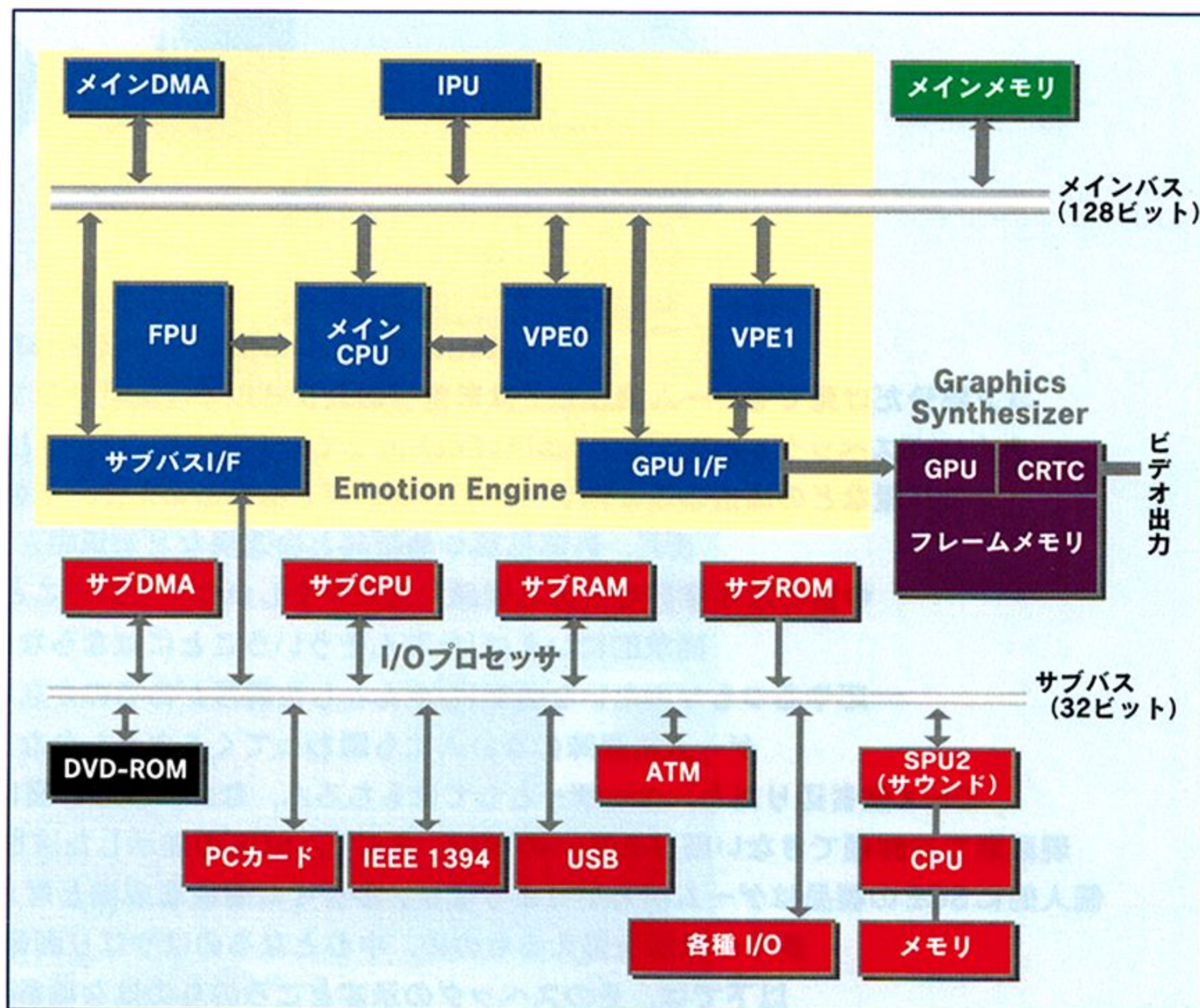



図1　PlayStation2の予想ブロックダイアグラム

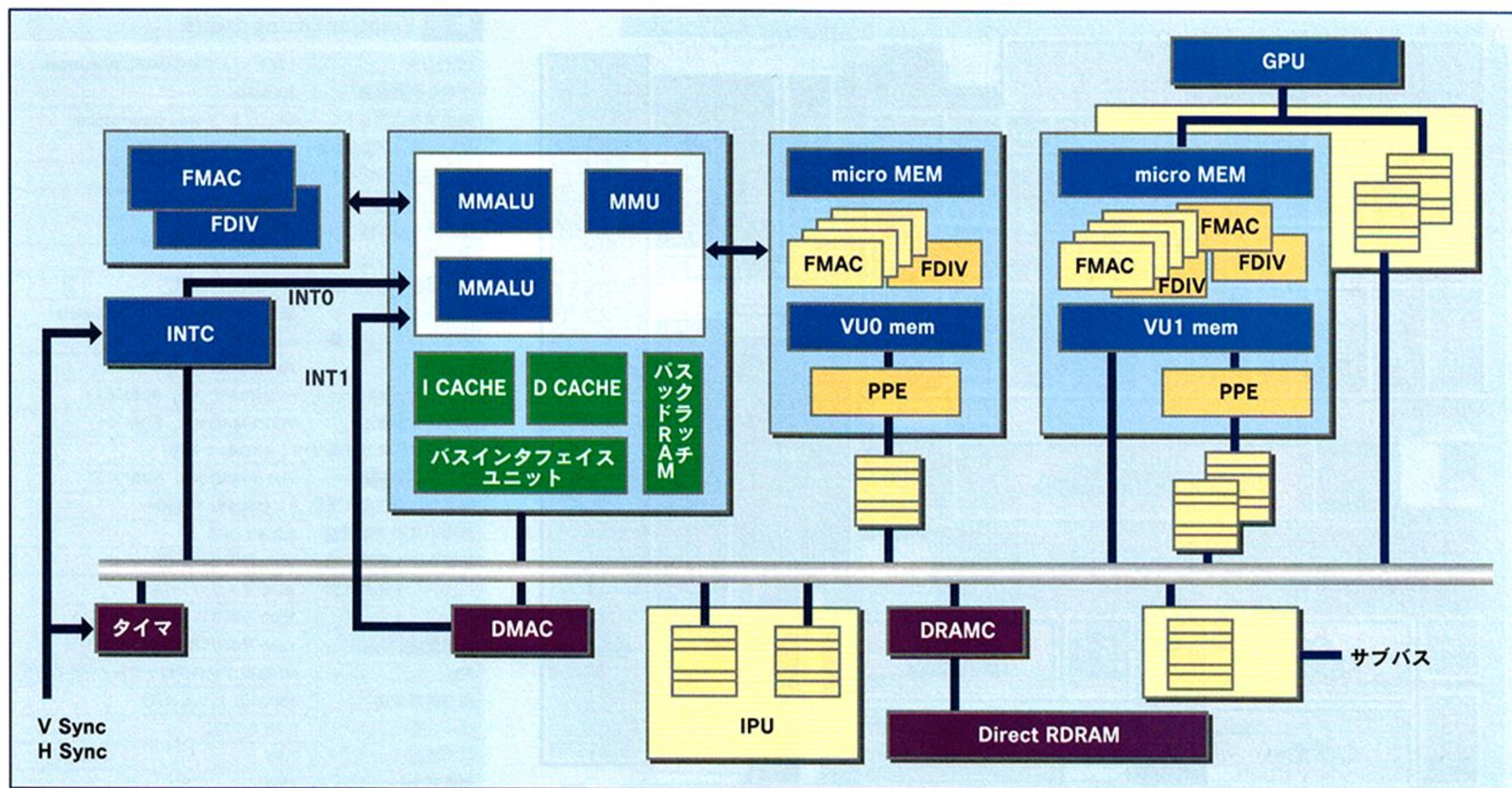



図2　EmotionEngineの内部ブロック図(予想)

と「32×128bit Reg File」とか書いてあるし，あちこちの線は「128bit」となってて本当に128ビットプロセッサっぽい。しかし，通常CPUのもっとも核となる整数演算ユニットは64ビット。確かに128ビット整数などを使ってもしかたないのだが，これで128ビットプロセッサと呼んでいいのだろうか……。128ビットレジスタがどのように使われるのかなど，いまひとつ把握しきれない。

プロセッサの300MHz動作はSCEから要求された絶対条件だったと東芝の斉藤光男システムLSI技術研究所長は3月25日付の日経産業新聞で語っている。SGI用のメディアプロセッサやMpactプロセッサを中止し，総力を挙げて製作したものらしい。現段階で300MHzを実現できるMIPS系のコアは限られており，Quantum Effect DesignのRM7000辺りがベースとなったのではないかといわれていたのだが(64ビット整数2段)，全体が128ビット構成だとこれも怪しい。おそらく独自のものだろう。東芝自身，MIPS系のCPU開発では実績のある企業だけに，ほぼ新規開発ということも考えられなくはないが，MIPS自体が128ビットデザインを出していない状況でこんなものを作ってまずくはないのかと心配になってしまう。

浮動小数点演算ユニットはチップ写真を見る限り，かなり単純化されており，VU0などのFMACと変わらない。単精度サポートのみになっているのではないかと思われる。その分，除算器を備えるなど，コア部分だけ見ても通常のMIPS RISCプロセッサそのままの構成ではなく，かなりカスタマイズが施されていることがわかる。スクラッチパッドRAMなどもほかでは見られない特徴だ。表2のように，分岐関係同士の命令以外はすべての組み合わせで2命令同時発行が可能とされており，推定IPC値は1.0程度，363～500MIPS程度の性能とされている(MIPS RISCに詳しい中森氏に，なんで2ウェイスーパースカラなのに1クロックで1命令がやっとなのかと聞いたところ，「そりゃあインタロックがあるからですよ」とのことだった。MIPSなのに……実際，結構止まってるもんらしい)。

CPUコアが演算するのが64ビット整数と32ビット浮動小数点数だと仮定すると，128ビットデータというのはコプロセッサアクセスのためだけに必要なのだろうか？

製造プロセスは4層メタル0.18μと発表されているが，0.18μm transistorという表記から，線幅自体は0.25μ，ゲート幅0.18μで設計されているのではないかと推測される。ダイサイズもまだ少し大きく，製品時(本格量産時？)にはさらにシュリンクされたパッケージで登場することになるのだろう。

問題となるのはVPUの部分だろうか。浮動小数点演算ユニットVU0とVU1が半直列状に接続され，1頂点あたりの3次元変換と透視変換を2クロックで終了する。構成を見た感じではスループット1クロックでいけそうな感じなのだが(実際，1クロックで処理すると記述されている資料もあるようだが，最大頂点変換性能はどれを見てもクロック値の半分となっている)，「回路規模が大きいのでどこかで妥協したんではないですか(中森)」といわれると，そんなもんかもしれないと思ってしまう。真相は不明。

浮動小数点演算器がこれだけ並んでいるのは壮観だが，頂点演算以外のことにどの程度使えるのかは不明。本来汎用であることは間違いないが，基本構成自体は3次元処理しか考えてないようにも思われる。出力はほっとくとGPUインタフェイスからGPU直行という経路ができあがっており，物理演算などで使えるものなのだろうかという点で若干疑問も残る。3次元処理以外にベジエ曲面からのポリゴン生成を行うということだが……。

ちなみに，最初にCPUブロックを見たときには，CPUコアのFPUとは別にFMAC(浮動小数点積和演算器)とFDIV(浮動小数点除算器)がCPUコア側に増設されているのかと思っていたのだが，どうも倍精度のFPU自体はなしで，VU0/1で使ってるユニットと同等なものを利用するようだ。VU0/1が使えない場合，物理演算などをここだけでこなすのは少しつらいかもしれない。

CPU本体ではないが，DRAMコントローラなどについても見てみよう。RAMはDirect RDRAM

I：命令アドレスフェッチ　A：実行
Q：命令フェッチ　D：メモリアクセス
R：レジスタフェッチ　W：ライトバック

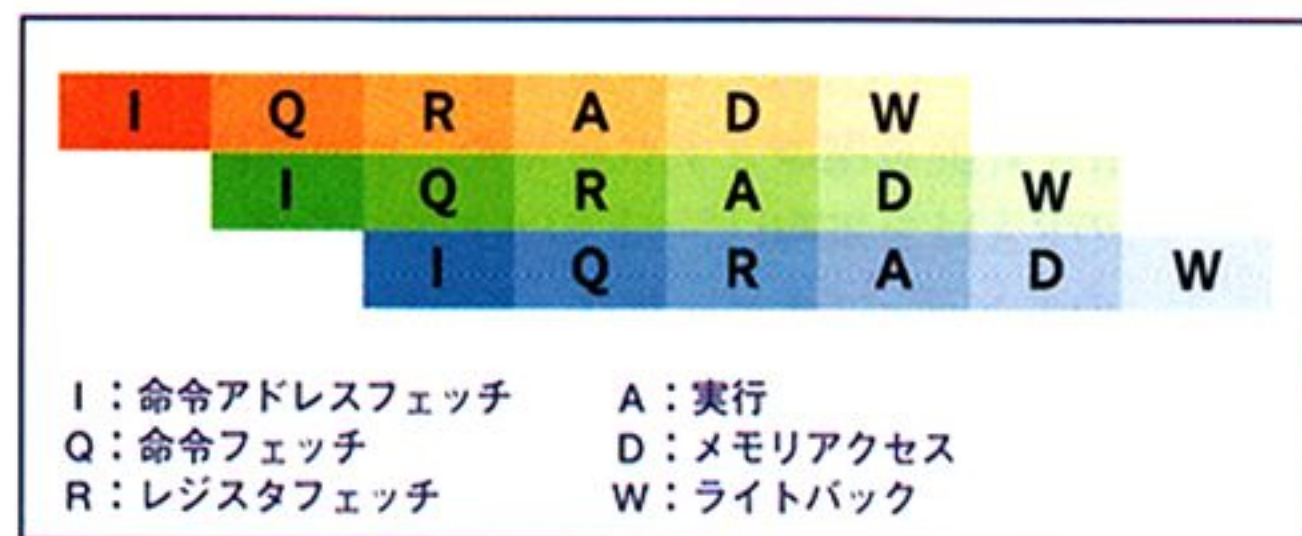



図3　命令実行パイプライン

表2　命令の同時発行可能な組み合わせ

	ALU	MAC0	MMI	Branch	COP1 Oper	COP2 Oper
ALU	○	○	○	○	○	○
MAC1	○	○	○	○	○	○
Ld/St	○	○	○	○	○	○
Sync	○	○	○	○	○	○
Branch	○	○	○	×	○	○
ERET	○	○	○	×	○	○
COP0 id/mov	○	○	○	○	○	○
COP1 id/mov	○	○	○	○	○	○
COP2 id/mov	○	○	○	○	○	○

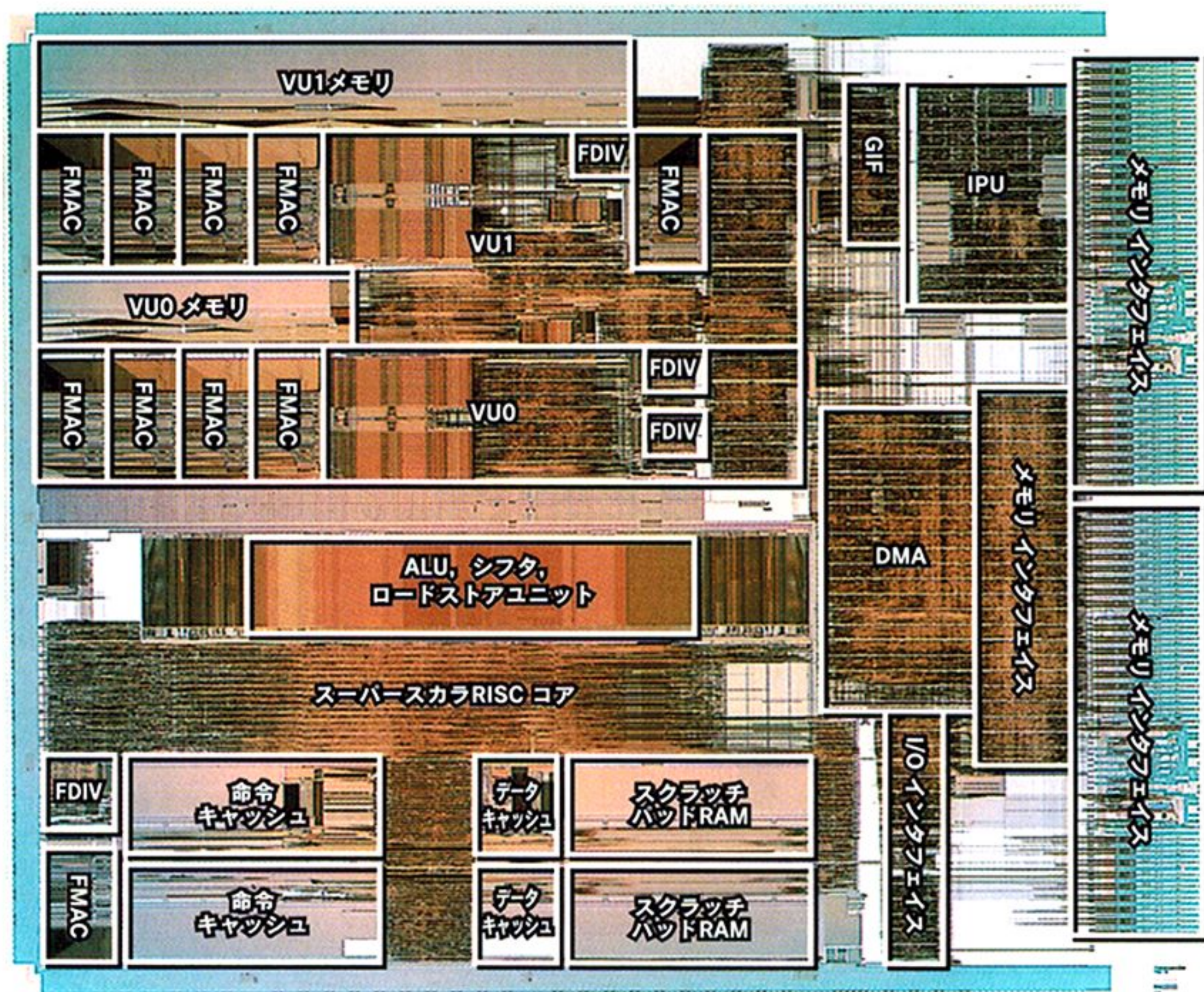



図4　EmotionEngineの内部ブロック

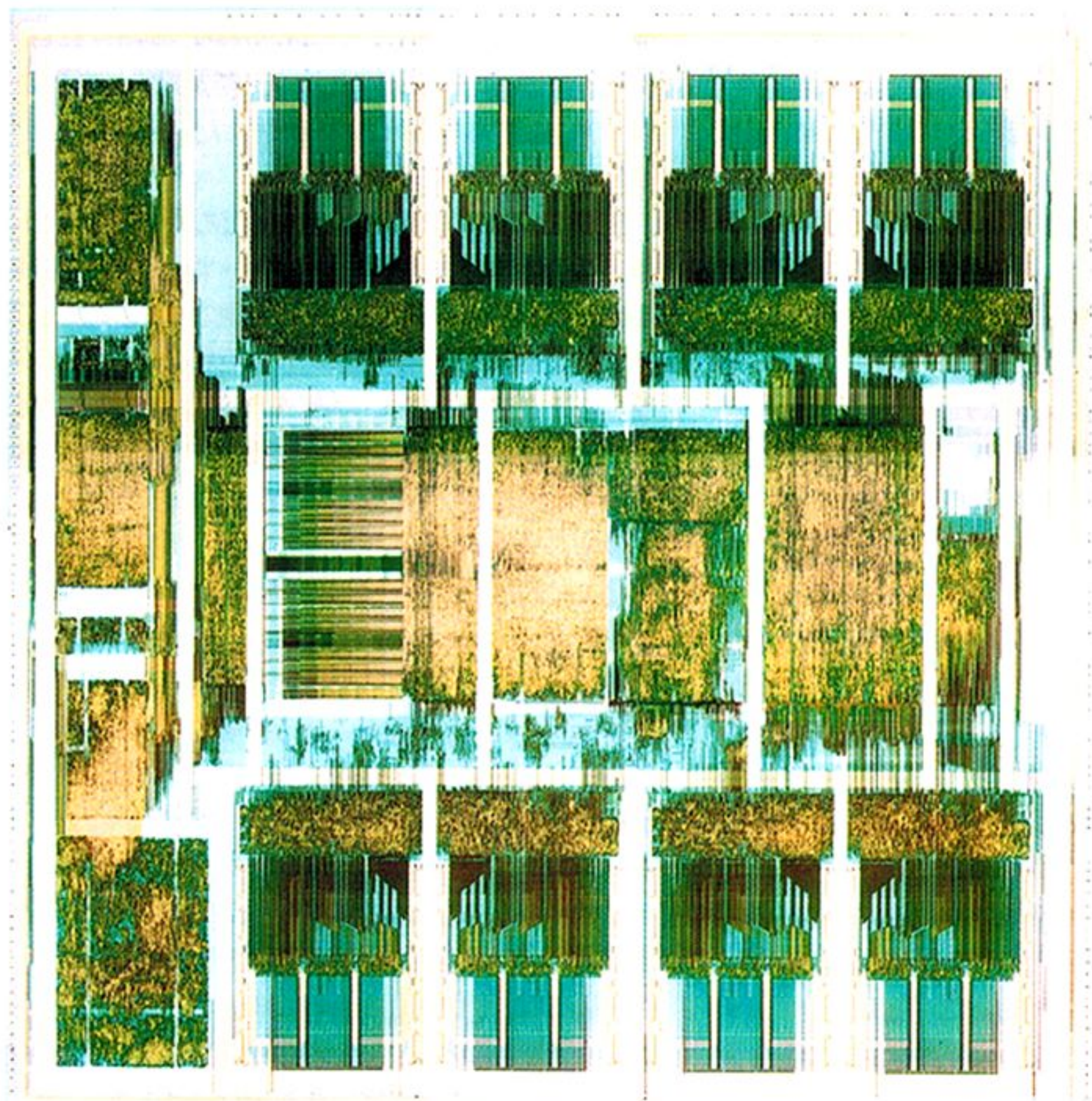



図5　GraphicsSynthesizer。真ん中がRAMで上下に3Dエンジン，左がCRTCなんだろうなあ。RAMDACは下かなあ……

表3　EmotionEnzineの仕様

項目	仕様
CPUコア	128ビット RISC（MIPS IV-subset）
クロック周波数	300MHz
整数演算ユニット	64ビット（2-way superscalar）
マルチメディア拡張命令	128ビット×107種類
GPR（整数レジスタ）	128ビット×32本
TLB	48ダブルエントリー
命令キャッシュ	16KB（2-way）
データキャッシュ	8KB（2-way）
スクラッチパッドRAM	16KB（dual-port）
メインメモリ	32MB（RDRAM×2ch@400MHz）
メモリバスバンド幅	3.2GB/秒
DMA	10-channel
コプロセッサ1	FPU（FMAC×1，FDIV×1）
コプロセッサ2	VU0（FMAC×4，FDIV×1）
マイクロ命令用メモリ	I：4KB/D：4KB
ベクトル演算器	VU1（FMAC×5，FDIV×2）
マイクロ命令用メモリ	I：16KB/D：16KB
浮動小数点演算性能	6.2GFLOPS
座標変換＋透視変換	6600万ポリゴン/秒
＋光源計算	3800万ポリゴン/秒
＋フォグ	3600万ポリゴン/秒
曲面生成（ベジエ）	1600万ポリゴン/秒
IPU	MPEG2 マクロブロックレイヤデコーダ
画像処理速度	15000万 ピクセル/秒
ゲート長	0.18ミクロン
コア電圧	1.8V
消費電力	15W
メタル配線層数	4
総トランジスタ数	10.5M Tr.
ダイサイズ	240mm^2
パッケージ	540ピンPBGA

が2チャンネル使用されている。DirectRDRAMは次世代の主力メモリとなるもので，PCではたぶん来年辺りから使われるようになるだろう。NINTENDO64でも使われていた普通のRambus DRAMよりもランダムアクセス性能に優れた構成の未来型メモリである。16ビットのバス幅で最大400MHzもの高クロック駆動に対応し，ダブルエッジドライブで2倍の転送量でプロトコル転送を行う。だいたいの感じをわかりやすくいえば，SCSIに似ているメモリバスだ。

400MHz駆動という公式発表で3.2GB/Sの転送帯域幅となる。ただ，現状ではDirectRDRAMの400MHz駆動はかなり厳しいといわれているのだが……。インテルの次期チップセットCaminoはこれがクリアできずに順延されている状況であると聞く。それに先駆けて400MHzで動作させているのなら（しかも2チャンネル），十分に偉業といっていいだろう。

IPUはMPEG2デコードのためのもの。かつてのMDECでMotionJPEGの展開を行っていたのがちょっと進化した感じだ。

スーパースカラCPUコアをはじめ，VPU，DMA，IPUなどが詰め込まれたチップがEmotion Engineである。ただ，Emotionを出せるといっても，頂点の補間くらいならたいした処理ではないんだが……。

●GraphicsSynthesizer

GPU＋CRTCにVRAMを内蔵した東芝のRAM混載型コントローラだ。チップ内の4MBのRAMを2560ビットのバスでつなぎ，16並列のエンジンで描画を行う。ある程度インテリジェントになっていて簡単な描画制御は自動でできるようだ。

バス使用の割り当ては，

書き込み1024ビット
読み込み1024ビット
テクスチャ512ビット

となっており，64ビットエンジンが16個並列して動作できるようにと1024ビットバスが用意されていることがわかる。RAMの統合により，コントローラ内部だけで超高速処理を実現しているわけだ。

ところで，PlayStation2のボトルネックは表示能力の低さだと書いたら正気を疑われるだろうか？

VRAM仕様は，RGB α 32ビット＋Z32ビッ

表4 GraphicsSynthesizerの仕様

GSコア	DRAM内蔵並列描画プロセッサ
クロック周波数	150MHz
ピクセルエンジン数	16並列
混載DRAM容量	4MB@150MHz
総メモリバンド幅	48GB/秒
内部総データバス幅	2560ビット
Read	1024ビット
Write	1024ビット
Texture	512ビット
最大表示色数	32ビット(RGBA：各8ビット)
Zバッファ	32ビット
画像処理機能	Texture Mapping/Bump Mapping
	Fogging，Alpha Blending
	Bi-/Trilinear Filtering
	MIPMAP，Anti-aliasing
	Multi-pass Rendering
画像出力	フォーマットNTSC/PAL
DTV	
VESA	最大1280×1024ドット
プロセス	0.25 μ
総トランジスタ数	4300万トランジスタ
ダイサイズ	279mm²
パッケージ	384ピンBGA
描画性能	
ピクセルフィルレート	24億ピクセル/秒(Z，A，T)
	12億ピクセル/秒(Z，A，T)
パーティクル描画性能	1億5000万個/秒
ポリゴン描画性能	7500万個/秒(微小ポリゴン)
	5000万個/秒(48pix四角形，Z，A)
	3000万個/秒(50pix三角形，Z，A)
	2500万個/秒(48pix四角形，Z，A，T)
スプライト描画性能	6600万ポリゴン/秒

トという64ビット構成とされているが，一方で最大解像度は1280×1024ドットとされていることからも(これだと4MB RAMでは24ビットフレームバッファしか取れない)，多少は柔軟な構成になっていると考えられる。ただ，メモリ容量からして，1280×1024ドット時にはバックバッファなどは取れないのでゲーム用途としてはあまり現実的ではない。それどころか4MBでは800×600ドットの64ビット時に3.7MBを1画面の表示のためだけに消費するということで，この程度の高解像度表示もあまり現実的ではない。これらの場合，まともなゲームをしようとすると垂直帰線期間内に描画を終了しなければならない(できてしまうかもしれないんで嫌になるが)。

出力媒体としてテレビのことだけを考えているのだったら640×480ドットでなんとかなるのだが，それではあまりに志が低い。

ちなみに，640×480ドット時でも1画面は2.3MBになるので普通に考えるとバックバッファは取れないのだが，これはパソコン並みにノンインタレースで表示したときの話。テレビの60fps表示だと640×240ドット表示を引き伸ばして使ってやればいいだけなので，1画面あたりのフレームバッファは半分で済む。

ひょっとしてぎりぎり2画面取れる640×400ドットあたりが標準となるのだろうか。この場合でもテクスチャ容量はほとんどないのでメインメモリから取ってくることになる。これだとVRAMを統合したり512ビットというテクスチャ専用バスをつけている意味がほとんどなくなる。

よって，推測できることは，

① 解像度は下げて使う
② 色数を落として使う
③ Zバッファは使わない
④ やっぱりテレビにしか出さない
⑤ 発表された数値が間違っている
⑥ ラスタ順に描画するなどで，裏画面を必要としない
⑦ 実はメモリ圧縮されている

となるのだが，さて？

⑥の条件ではマルチパスレンダリングをやるのがきつくなるので，あまりないと思っているのだが？　もしかしたらなにか上手い手があるのかもしれない。αを使わずに合成とかいう特許は提出されていたようだが……。

バックバッファなしというのはそれだけレンダリング速度に妥協が許されないということでもあり，常識的には選択しない方策だ。

いずれにせよ，インタレースVGAの状態でさえ，MIP-MAPテクスチャ，マルチプルテクスチャ，ハイトマップの設置，背景球への描画などを考えるとちょっと心もとない。4MBではせっかくのシステム性能を発揮できないのではないかと懸念される。この辺はPlayStation1の頃と事態が変わらないのかもしれない。

一応，「現行のTVのみならず将来の高精細デジタルTVにも対応した，映画に匹敵する高品位の三次元画像をリアルタイムに描画することを可能とします」(GraphicsSynthesizerプレスリリースより)ということにはなっている。いや，H-Sync取れるみたいだし，最大解像度でも256KBバッファが余ってて，テクスチャは外から取らなくても走査線から逃げ回って内部転送してやればいいんだから，できないとはいわないんだが……。

なお，メインRAMとは128ビット/150MHzのバスで接続されていると思われるので(2.4GB/S)，PCのAGP×4モードの倍以上の帯域は持っていることになる。パケット転送後に展開して使用というかたちになるので，AGPのDirect Memory Executionでの効率と考え合わせると性能評価は微妙なところかもしれない。

●周辺部分

サブバス方面はいわゆるPlayStation1互換部だ。DVD-ROMがついたり，SPU(Sound Processing Unit)のバージョンが上がってたり，さらにサウンド用CPU(形式不明)が加わっていたり，IEEE1394やUSBといったものが加わっているものの，LSIロジック製R3000系I/Oコントローラが受け持つことになる。従来型PlayStationのソフトはこの部分が実行して互換性を保つことになる。公式には，速度などを含めてまったく同じということになっているのだが，キャッシュを拡張したという記述もある。従来型PlayStationでキャッシュが増えると，速度もかなり変わる可能性があるのだが，実際にどうなるのかは実機が出るまでのおあずけだ。

サウンドCPUはまったく話が出てこないのだが，東芝RISCだとするとTX49系というのが妥当か？　周辺担当のLSIロジック製ということもありうるが，どちらにしてもMIPS系なのだろうか。

IEEE1394を介して相互のメモリ内で高速転送をするとか，ネットワークへ接続するとか，非常に幅広い拡張が示唆されているのも興味深いが，実際にどのようなかたちで出てくるのかは予想ができない。

特にPCカードスロットがついているというのが気になるところだ。PC屋さんならよく知っていると思うが，これはかなりかさばって，しかもコストの高い部品である。あれば非常に便利なのだが，用途が広がりすぎることは歓迎されていないのかもしれない。USBなどがあれば，さほど必要なものではないという意見もある。ちゃんと搭載されるのかいまひとつ不安が残る。一部では，PCカードといいながら，実はソニーのメモリスティックスロットに落ち着くのではないかとかいう推測もある。さすがに，これだけPCカードをつけると公言しているのだから大丈夫だとは思うのだが……。

周辺機器はPlayStationのものが使えるということになっている。この辺が普通のゲームメーカーとまったく発想の違うところで，一般のメーカーだと周辺機器の売り上げも結構おいしい収入源なので，普通は絶対に互換性は持たせない。ただコントローラなどはPlug&Play式のものになると思われるので，新しい仕様の製品も登場してくるのだろう。

接続されているコントローラの色を自動判別するとかいう特許が出されていたので，コントローラをはじめ，本体色などにバリエーションがあることも考えられる。

そのほか，周辺関係ではDVD-ROMでDVDが再生できるのかどうか，ピックアップは1個なのかとか，実機を見るまではわからない部分が多い。

SCEが特許庁に提出していた昔の資料を見ると，映像入力，音声入力などを想定していた可能性がある。もともとPlayStation自体が映像機器からスピンアウトしてできてきたようなものだから，映像系の装備や処理が考慮されていてもおかしくないのかもしれない。現状では映像入力などは残っていないのだが，IEEE1394で代替されたものとも考えられなくはない。PlayStation2自体，ゲーム機というだけでなく，どういう用途で提案されてくるのかまだまだ未知数である。この辺は今後の展開が特に怪しい部分でもある。

●DMA

主にCPU部分に関わる話なのだが，DMAについては別個に話をしておこう。資料では10チャンネルのDMAが装備されるとなっている。

さて，これらではCPU内に設けられた超高速メモリであるスクラッチパッドRAM，VPU0内のワークメモリ，VPU1内のワークメモリ，GPU内のメモリの4カ所へのデータの超高速転送が可能になっていると思われる。

データバスは128ビットで接続されておりバンド幅に問題はない。しかしコマンドなどの細かいデータや，個別のデータを送っていると広いバスを使っていても無駄が多くなる。そこで，データはパケット化されてまとめて転送されるらしい。その際にデータに加えてプログラムなども付加され，パケット自体がプログラマブルなかたちで転

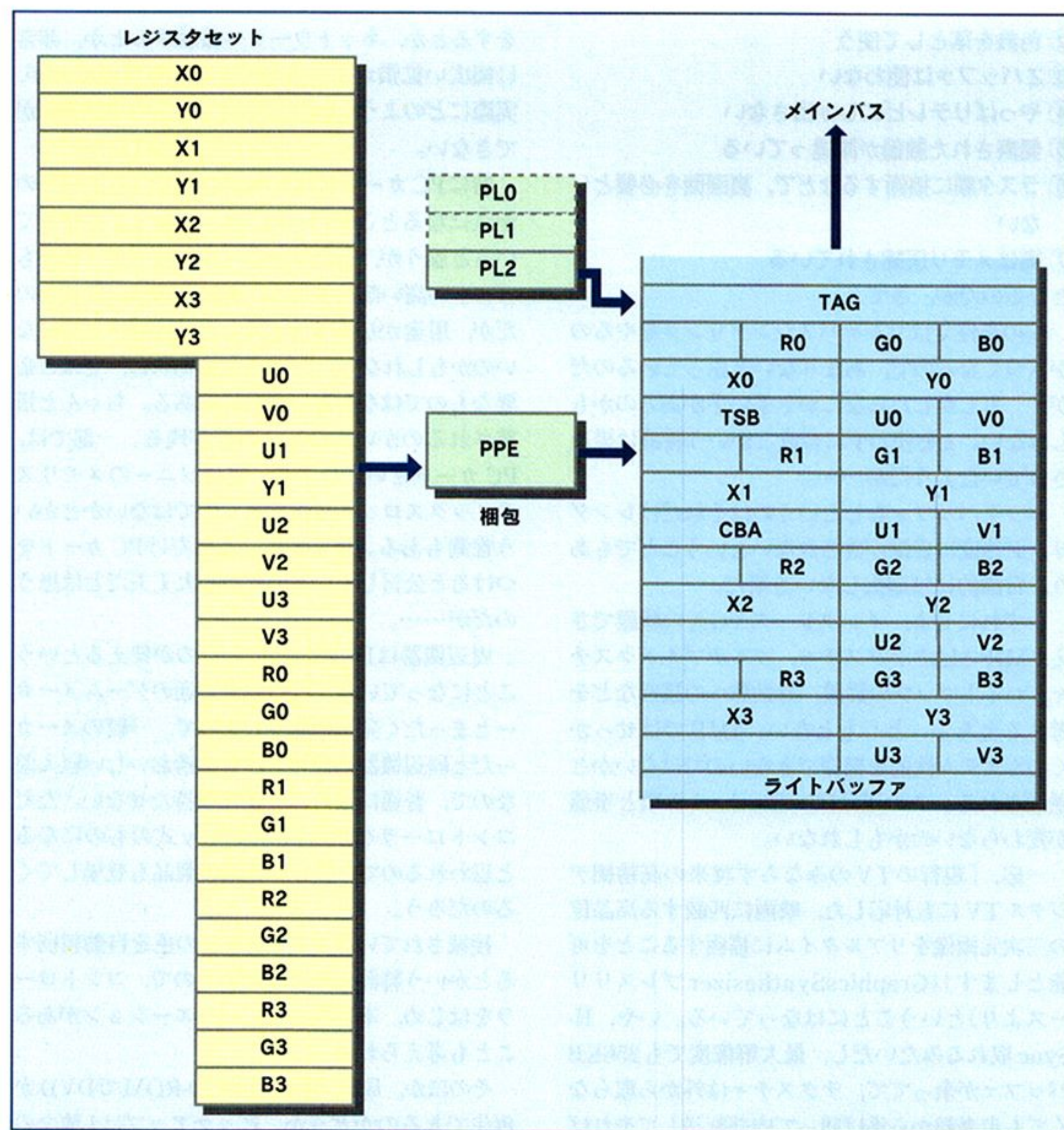



図6 DMAデータのパケット化動作の模式図(SCE提出の公開特許資料より)

送されるようになっている。バスを常に最適なかたちで利用するためにこのような機構が採用されていると思われる。次のユニットに送るときに，そのユニットが利用しやすい形式にパケットを作って転送してやる。パケットはキューに溜められ，プログラマブルパケットエンジンで展開され処理に回される。処理後に必要があればまた次のユニットが使う形式でパケット化され，次々と転送される。バスの使用効率という面では有効だが，実際のパフォーマンスでどれだけ差が出るのかはちょっとわからない。

私見

個人的な印象から先に述べておこう。3月2日の時点で発表された資料は別段驚くほどのものではなかった。すでにCPUについては概要が発表されていたので，おおよその性能は予想の範囲内だった。とはいえ，CPUよりもGPUのほうが格段に凄いらしいという噂は耳にしていたので，GraphicsSynthesizerのほうが気になっていたのだが，確かにいくつかの点で驚かされたのも事実だ。半面の疑問点もすでに挙げた。

PlayStation2の概要について，初めて準公式な情報が明らかになったのは，マイクロソフトの3D APIの囲い込みに対してEE Timesが行ったSCE現社長の久多良木氏のインタビューが最初だろう('98年7月13日)。

http://www.techweb.com/wire/story/TWB19980713S0005から引用すると，

```
Sony' s Kutaragi said a Sony Computer
Entertainment engineering team based in Tokyo
is working on a whole new generation of real-
time image-rendering technologies, from
silicon to platform algorithms to software
titles, for the next PlayStation. "Today's
video game computer graphics look like
computer graphics," he said. "Our goal is a
film-like graphics quality that won't make
viewers conscious of or annoyed [by the fact]
that they are indeed looking at computer
graphics."

One generational leap on which Sony is
focusing, Kutaragi said, is synthesis of
" 'emotion' in characters rendered in real
time." To date, animators have been charged
the task of imbuing game characters with an
admittedly limited emotional range. Sony's
"dream is to automate the process and
synthesize it in real time on a game
platform."
```

前回のDreamcastのときも参考にした記事なので，内容を翻訳するまでもないだろう。

いいたいことはなんとなくわかるのだが，実際にこれを読んだときは「おっさん，寝ぼけてんじゃねーよ」ってのが素朴な感想だった。

プレステや業務用のアーケード機にしても，リアルタイム3D処理として行われている内容はひどく原始的なことでしかない。表現力などの問題でいえば，CG業界で大昔にあったトピックが掘り出されているだけの話なのだ。CG業界の流行が何年かすればリアルタイムに実現できるようになるというのは確信を持っていたが，ゲームコンソールにまで落ちてくるのにだいたい5年と踏んでいた。しかし，ここで述べられていることは，最先端のCG業界でもまだ無理なことなのだ。

当初20年分くらいあった差が急速に縮まってきているのは確かだが，まだ追い抜けるわけがない。なぜならこれはハードウェアの技術ではなくてソフトウェアの問題だからだ。一方で，SCEのソフトウェア技術者がどの程度の力を持っているかというのも，だいたいわかってたので，単純に無体な話だと思った。

あと，ちょっと次世代ゲームの認識で追加説明が必要かもしれないので補足しておこう。久多良木氏の述べていた方向性は別に間違っているとは思わない。ただ，私は次の次の世代の話だと思っているだけだ。これはいまでも変わらない。次の世代ではゲームの作り方が完全にオブジェクト指向にシフトしつつ，物理演算モデルがキーとなるというのは，まあ，業界の人なら誰だって理解していたことだろう。

まだちょっとわかりにくい人もいるかもしれないので，なにが論点になっているかをはっきりさせておこう。

たとえば，綺麗な静止画像があったとして，それだけでゲームのビジュアルは十分だという事態もあるかもしれない。これを仮に自由度1の状態としておく。たとえば，取り込み画像が使えればそれで十分とかいう場合だ。

これだと寂しいのでアニメーションをつけよう。これを自由度2とする。アニメーションやムービー再生で画面はずいぶん派手になる。

ムービーだと容量でパターンが限られるので，映像生成をリアルタイム3DCGで置き換えよう。最大のメリットは「視点を変更できる」ということだ。これを自由度3とする。動きはモーションキャプチャなどでリアルな映像を実現できる。

モーションが固定されていると展開が単調で容量も食うので，モーション自体を自動生成してやろう。これを自由度4とする。

さらに，細かい指示をしなくてもオブジェクトが自分で動いてくれると助かる。これを自由度5，AIまでいけば6と定義しよう。

重要なのはこれらのどれもが，「実写並み」の画質は実現できるということだ。

さて，PlayStation2のデモ映像としてニュースに流れていた鉄拳とか永瀬麗子とかFF8あたりは自由度3のレベルの話でしかない。まるっきりのモーション垂れ流しではないのだろうが(リアルタイムボーン制御とかそれなりに大変そうな気もするがPlayStationやSATURNですでに行われていたことだ)，たいていはDreamcastでもやれなくはないんじゃないかと思われる。ゲームっぽいデモはなにを見せたいのかいまひとつわからない。たくさんのオブジェクトが動かせる……ったっていまさらだしなあ。

さすがにSCEの用意したデモはポイントを押

図7　火花。パーティクル＋モーションブラー

図8　毛玉。3Dライン＋アンチエイリアス

図9　意図不明。相互の写り込みや影落としくらいすればいいのに

図10　風祭り。さあ，被写界深度設定されてるか？

図11　羽毛。モーションを見てないんで物理演算かどうかは不明

さえているものが多いみたいだが，実験なのか，具体的な提案なのかはっきりしないものもある。CG屋さんはいろいろやってたみたいだけど，あまり理解してもらってないだろうなあ。

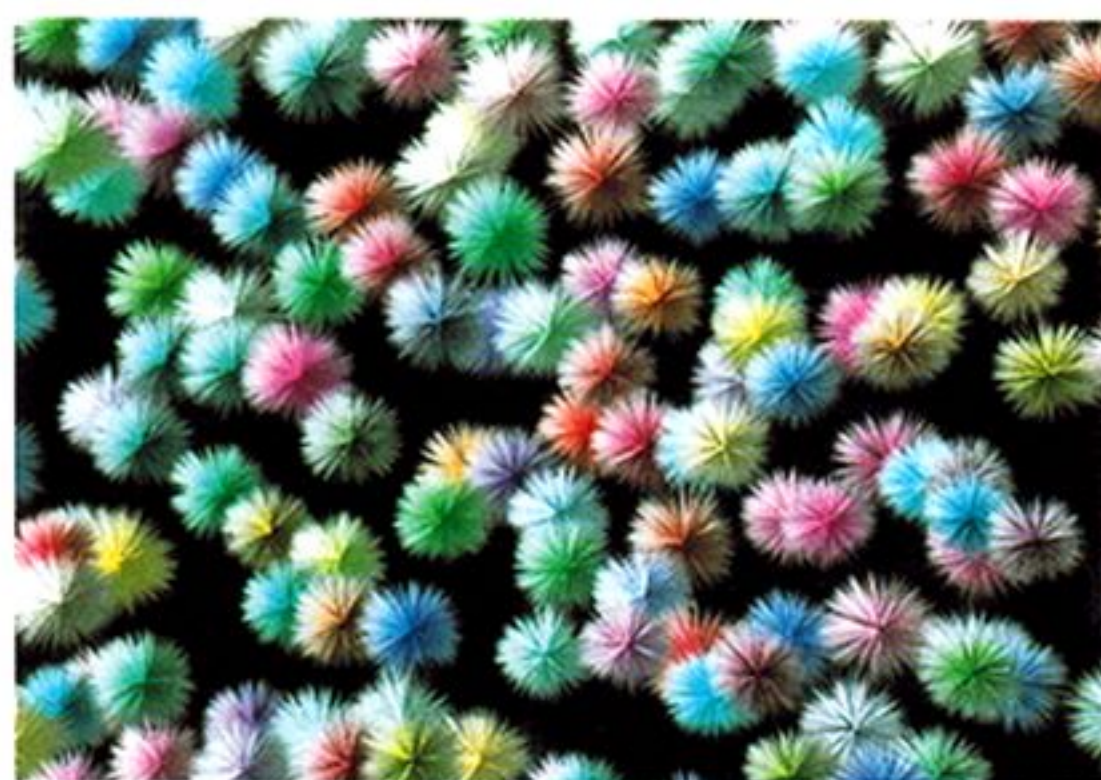
デモ解説

画面写真なしでいこうと思ったのだが，さすがに無理があるので以下の図はSCEの公式資料と「週刊ザ・プレイステーション3月26日号」よりの引用だ。そこでないものはしょうがない，ということで，一気に解説してみる。

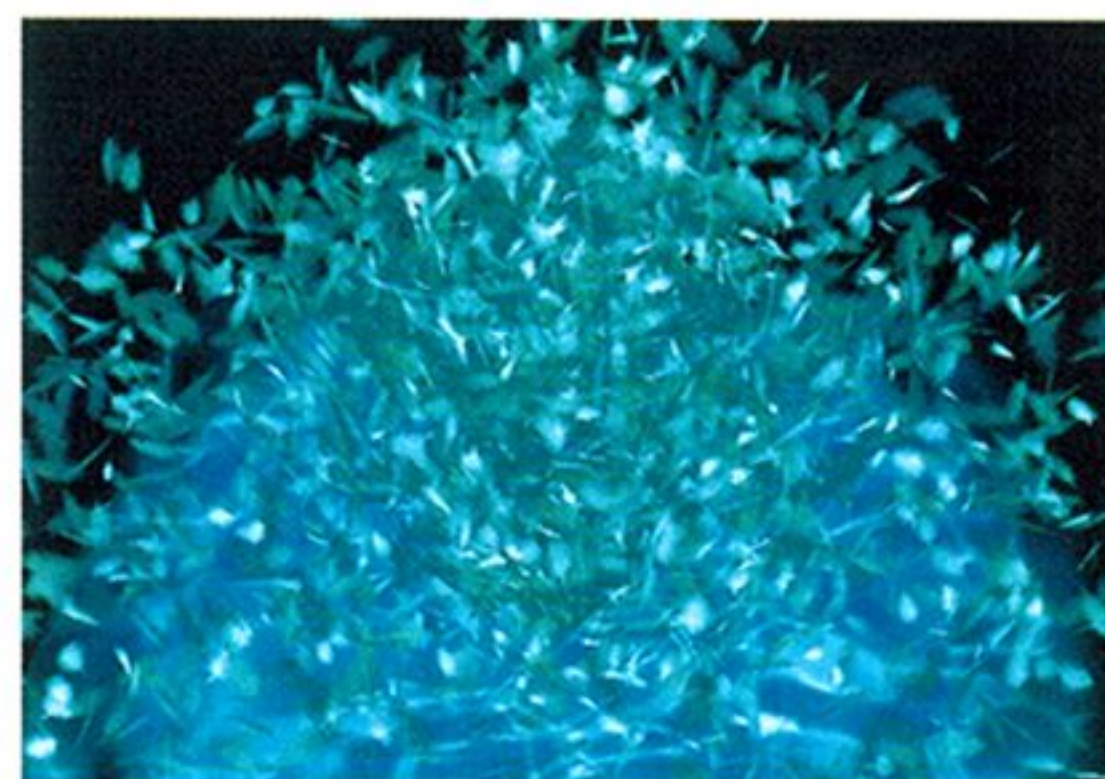
パーティクル。火花が散ったり，花火だったり，火が燃えていたりというやつ。すでに全然新しいものではない。PCゲームだと当たり前で使われている。ミドルウェアのデモだろう。

3Dライン。毛玉とかFFムービーのおっさんの眉とか。10年近く（以上？）前だが，CG業界ではいろいろなマテリアル表現追究に余念のない時期があった。木や石やガラス，プラスチック，金属，なんだかよくわからないものなど……さまざまな材質のティーカップが画面でくるくる回っていたものだ。なぜティーカップなのだかはよくわからないが，一般的にティーカップが使われていた。それらの表現競争もひと段落した頃に出てきたのが「毛皮のティーカップ」だった。昔，丹君に聞いたところ「Zバッファ内に直接ライン引いてやればいいだけなんで，見た目ほど重い処理じゃないんですよ」といわれ，なるほどと思ったことがある。Zバッファ採用の機種ならすべてに実装可能だ。永瀬麗子の髪の毛あたりはポリゴンによるもの。というか，作り方は別ページに解説されているので参照のこと。

風祭り。ナムコがSIGGRAPHに出してた，ひらひら生命体のデモ。圧倒的なポリゴン数なので，目立つことは目立つが玄人ウケなためか，ナムコものだと永瀬麗子しか写真がないなあ……。画質の悪いMPEGデータでしか確認できなかった。NICOGRAPHではユーザーが操作できるデモをやってたと思ったので，リアルタイム表示自体はワークステーションクラスのマシンでは可能なのだろう。このデモの核心はモーションブラーと被写界深度設定なのだが，いまひとつ確認できず。たぶんやってると思われる。

影。ざっと見て，はっきりした影はバンディクーとGTくらいにしかついてなかったようだ。どちらも地面のみへの嘘影と思われる。オブジェクトへの投影というものは確認できなかった。処理としてちゃんとやってるのはFFのおっさんの顔だけ。このクラスの映像になると影がないと不自然というか，ゲーム機の絵にしかならない。いちばん懸念される部分でもある。

で，そのFFムービーのおっさん。画像的にはもっともPlayStation2にふさわしいもの。FF The Movieで有名なおっさんの顔をアニメーションさせている。ほぼそのまま動いているのはとりあえず立派。モデリングデータは映画のものからポリゴン化したものだろうし，マッピングもちゃんとバンプまでやっている。そして，ちゃんとした影をつけているという点がもっとも評価できる。というか，これくらい作り込んだモデルでは影くらいつけないとかえって不自然だろう。さすがに3DCGというものを理解しているなあと思わせる。「影つけて，バンプやって，アンチエイリアスしなくてどうする！」というのは実はDreamcastのデモを見たときに思ったことだ。誰も次世代機に多くのものを求めていないのではないかと心配していたのだが，ちゃんと性能を引き出してる人もいるのでとりあえずはひと安心か。

このデモにはロードメーターがついている。見ると頂点数75857で，EmotionEngineの負荷値は32%，GraphicsSynthesizerの負荷は50%となっている。現状のハードウェアが250MHz駆動のもので製品時には300MHz化されるとはいうものの，ちょっと負荷が高いような気がする。この程度の処理が当たり前になってもらわないと困るのだが……。画像の大きさで頂点数が変化していないところを見ると，モデルデータは生頂点のポリゴンデータで与えられている。PlayStation2では曲面（NURBSだと思うのだが，ベジエと書いてあるなあ……）のデータのまま与えて頂点生成はハードウェアで行うというのが筋である。

演算能力からすると，1画面あたり26万頂点分となる。パラメータで画質は調整できたり，負荷に応じて頂点数を増減させるような処理も当然組み込んでくるだろうからあまり心配はいらないのだろうが，先ほどの7万頂点のデータでは小さな写真で見ても拡大時にポリゴンエッジが見えている。この辺りの最適分割をうまくやらないとせっかくの機構が使いものにならないこともありうるのだが……さて？　もちろん対処法はいくらでもある。どういう風にやってくるかが楽しみだ。

GT。かなり前に横内君と話をしたときにモーションブラーは必須とかいってたので次は確実にやってくると思っていたが，割と単純な方法で実装してきた。ざっと残像を数えて8以上あったので16段階の画像合成だと思ってたのだが，10段階だそうだ。残像はやや離散的すぎる気がするが，やり方自体はセオリーそのままなんでしかたないか。

が，映像を見ると映り込みが整然としすぎていてつまらない。作ってる側はよくわかっていると思うが，たぶんGTのクルマデータを使い回しているだけなので，ポリゴン数が少なすぎるのだろう。この部分だけ見ればGT1のほうがデキがいい。演算精度とモデリングデータのバランスが悪いので，もっとアンジュレーションを含んだモデリングデータを作り込むか，環境マッピング時のソースアドレスにモジュレーションを加えるなどの処置が必要かもしれない。ライトがボディ上にうねうねうねっと映り込んでいかないくらいなら，はっきりいってやめたほうがいい。

流し台ほか。物理モデルってのは，物体のモーションなどをシミュレーションで生成しようという試みのためのものだ。とりあえず，いくつかはちゃんとこういうデモがあるというのは十分に評価できる。たとえば，石という物体を投げて，石が池に落ちれば，その運動エネルギーの分だけ音が発生し，波紋が立ち，石は沈み，エネルギーの拡散が集束していく。

十分な雛型が用意されていれば，細かいことは

図12　GT.モーションブラー。相互写り込みは確認できず

図13　流し台。反射屈折，および物理演算系

図14　これがナムコの風祭りという作品の解説ページ

図15　PS2のデモに対するスクウェアの公式解説ページ

指定しなくても，オブジェクト同士がメッセージパッシングを繰り返して適切に動いてくれる。石を投げる，初速値や空気抵抗などをもとに軌道を決めつつ石が移動する。今度は窓に当たった。イベントが発生し，ガラスの強度と石の重量，エネルギー値が交換され，ネゴシエーションを取ったのち，それぞれ適切な処理に移行する。ガラスが割れた場合に石だったらどの方向に落ちるのか。ガラスの破片はどう飛び散るのか。この辺は確定論で片付けてよいので物理演算である程度まではシミュレートできる。多体問題に関しては三沢君の記事を参照のこと。関係の複雑な事象でもカオスなどを取り込んでそれなりの結果はたぶん出せるのだろう。数学屋さんに聞いてみないと詳しくはわからないんだけど。

こういった処理系を一度ちゃんと作っておけばいろいろ楽にはなるだろうし，メーカーが開発キットともに提供するのがいちばんよいのも確かではある(開発力のあるところはちゃんと自前で用意するだろうけど)。きちんと作り上げてクラスライブラリ化，ないしはROM化してハードに組み込むくらいのことが必要になる。うんざりするほど，しんどそうな作業だ。メモリも馬鹿食いしそうだ。そもそも，そんなことができるのか？という疑問がぬぐい去れない。

さらに，ゴールだというのはわかるとしても，人体ってのはテーマとして重いものがある。おまけに表情……とくればなおさらだ。それはまさに未来の姿である。こういうのが実現したときのプログラムはさながら映画のシナリオのようになるだろう。オブジェクトはモーションを内包しているだろうから，「歩け」といえば歩くのは当然だ。思考実験として，未来形のコーディングを想像してみよう。

・女(ベース東洋系B：変形はユーザーマトリクス24番を適用)
・上手(1.1，-0.017，-0.35)より，
・小走りで登場(小走り88：方向(-1,0,0)速度補正+0.13)
・右足をやや引きずりながら(関節31：自由度補正-0.2)

といった感じだろうか(もちろんテキストベースで書いていくのはナンセンスだろうが)。

しっかりした人体基礎オブジェクトが与えられていれば，必要なだけオーバーライドしてバリエーションは無数に作れる。かなり理想的な環境に近い。

なんとなく非現実的な話が多くなったようにも思われるが，自由度4。先ほど挙げたこのレベルの処理では当然に要求されてくることである。久多良木氏の発言には自由度4以上への展開が示唆されている。

永瀬麗子はCGツールのモーションやモーションキャプチャデータではなくて，自分で重心を管理，移動しながら歩かなければならない。モーションを切り替えたりモーフィングでは志が低い。自由度3から4への壁は非常に高い。3のままでいいなら，さほど多くの物理演算は必要とされない。現状の格闘ゲームに毛が生えたくらいで十分だ。現状のノウハウ，ハードスペックではその辺りが精一杯のはずだ。おそらく，その辺りに対する現実的な回答をシェンムー辺りが見せてくれるのだろう。なんのかんのいってもソフトメーカーとしてのセガは世界でも超一流なのだ。

歩く物体は正しく転ける必要もある。2足歩行ロボットの実現が結構難しいのと同様，ちゃんと作れば歩かせるだけでも大変なはずなのだ。簡単に歩くようだと，ちゃんと作ってないと断言してもいい。

ハードの性能が上がってポリゴンが増えてくるということは，曲面の多い，柔らかい物体が増えてくるということだ。コリジョン時の変形や張力，外力やモーメントによる頂点の変動など，物理シミュレーションは困難を極めるだろう。すでに成果が上がっているのなら素晴らしいが，それは疑わしい。

ミドルウェアの展開

だいたい以上のようなことから，自由度4への展開は無理だろうと推察したのだが，SCEは意外なところでこれを迎え討った。海外ソフトハウスとの提携である。ミドルウェアとして3D STUDIO MAX用プラグインを作っていたAnimation Science，そしてRenderWareと手を組んだのだ。

よって3Dの基本システムはそのままRenderWareになるものと思われる。Windows3.1の頃から使ってた人にはずいぶん懐かしい名前かもしれない。RenderWareはキヤノンの子会社が作

図16

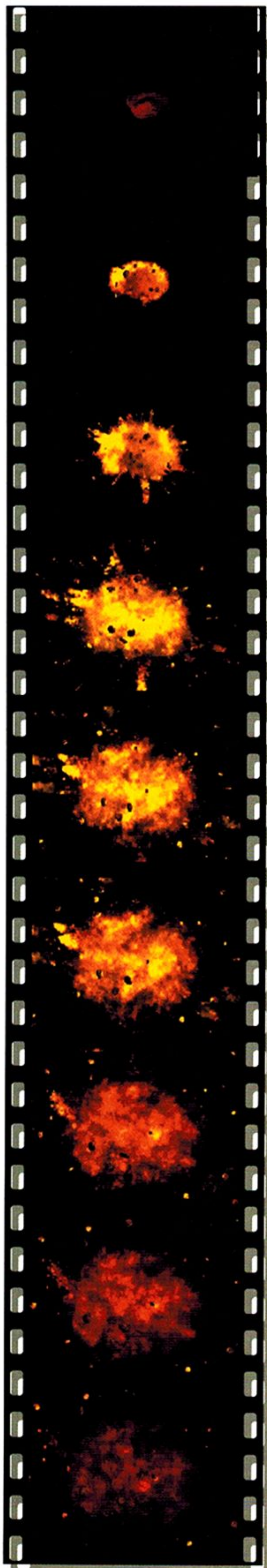



図17

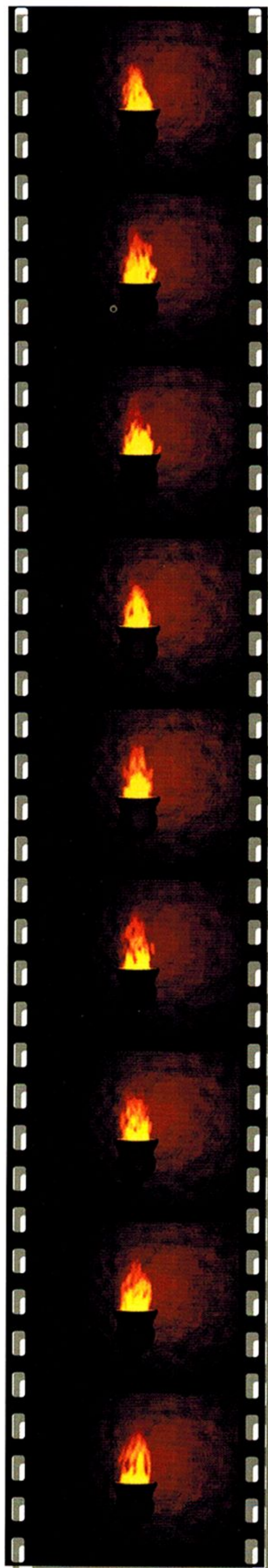



図18

図19

ったリアルタイム3Dシステムで，当時486マシンでも目を見張るような3D表示を実現していた。Direct3Dの構想が現れてからはついぞ名前を聞く機会も減っていたのだが，システムはバージョンアップされて健在であったらしい。

汎用3DAPIでハードウェアサポートのあるものは意外と少ない。Direct3D，OpenGL，Java 3D，RenderWare，BRender，QuickDraw3D，Mesa，Heidi，RealityLabくらいだ。RenderWareは高機能なのにシステムが非常に小さいということでもゲームコンソール向けに適しているようだ。確かにDirect3DやOpenGLはどう見ても小さなシステムではないし。

APIは固定されているのだろうから，PC上でRenderWareによる開発をしていれば，もしかしたらコードをほぼそのまま持っていけるのかもしれない。

比較的最近のPC用ソフトではRedlineRacerがRenderWareを使って作られている。DOS/Vmagazineのゲーム担当者に話を聞いてみるとゲームのほうの評価はさんざんだが，ビジュアルはそれなり。ただ最近はもっと凄いのが出てるので（ほとんど実写感覚），全体的にもう古いという評価だったが。

とにかく，CGツールなどを作っている技術をそのまま流用すれば，CGムービーに近いものをリアルタイムに生成することもできるだろう。というよりは，それをやるためには，プラグインツールでやってるノウハウを取り込む必要があったということかもしれないが。これなら現状の最先端に近いところの技術がそのまま入ってくる。確かに自社開発しなければならないというわけではないし，力量を考えれば餅は餅屋が当然なのかもしれない。

しかし，ここにきて，「映画並み」の画像を目指していたはずのPlayStation2が，いつのまにか「CG映画並み」の画像をリアルタイム生成するもの，という風にすげ替えられてしまっていることもわかる。

それでもAnimationScienceの製品を取り込んだというのは興味深い。RampageとOutBurstという3D STUDIO MAX用の2つのプラグインを見てみよう。これらは物理シミュレーション系，特にルールベースの群体のモーションコントロールとパーティクルなどの処理に長けているツールだ。

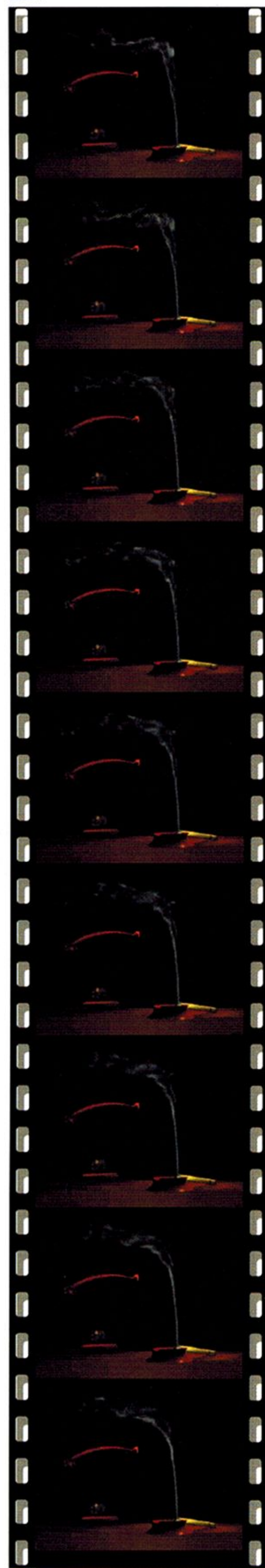
デモムービーからいくつかのアニメーション例を挙げておこう。従来はプログラマが制御していたようなものを自動制御するものが提供されるということになる。

SCEの直面している最大の問題は，まだ誰も実現してない処理だと外部からも取り込めないということかもしれない。

不安要素はあるか？

スペック面からちょっと見てみたい。カタログスペックだとしても，大半がはったりでもたいしたものであることは間違いないのであまり心配はしていないのだが，テクスチャを貼るだけでポリゴン性能が半分になっているのが目を引く（明ら

図20

図21

かに間違いっぽい数値は無視しよう。たぶんテクスチャつきだ)。

テクスチャ専用バスを設けながら，適切にパイプライン化されていないということだろうか？ α合成などはVRAMフォーマットにまとまっており，一括して行われるのは当然として，それ以外のフィーチャが軒並みマルチサイクルで行われるとちょっとつらいかもしれない。マルチテクスチャ，バンプマッピング，バイリニアフィルタリング，アンチエイリアス時のポリゴンレートだと最悪，ノンテクスチャ時の1/8くらいになってしまうのではないかと思われる。PS1の場合，テクスチャを貼って，グローシェーディングをかけると素直に1/3のポリゴン性能になっていたことを考えると，ありえない話ではない。それでも十分な値ではあるのだが，最近のPC系のビデオチップはもう少し利口で，シングルパスで数多くのフィーチャをサポートするようになってきている。

VRAM容量は置いておこう。次に，VRAMフォーマットだ。RGBα各8ビット＋Z32ビットという，ある意味，至極当たり前の構成であり，64ビット統一という綺麗な形式なのだが，はたしてそれで十分なのかという問題だ。具体的にいえば，ステンシルバッファがないことだ。プログラマブルに作ってあることも十分考えられるが，合成などはハードウェア処理なので固定である可能性も高い。

次世代コンソールに求められているものはなんだろうか？　画質は当然として，より高いポリゴンカウントなどが本当に求められているのかどうかは正直疑問を持っている。

個人的に次世代機で要求される機能として想定しているものに影がある。影，フルスクリーンアンチエイリアス，立体映像，フィードバックコントローラ，そしてネットワークだ。後者3つはコストとハード的な問題が大きいので置いておくとして，ポリゴン映像がもっとも不自然に感じられる要因となるのは，陰影の画質の不自然さだ。これは近づいてポリゴンがカクカクしているとかいう問題よりもずっと大きい。

また，ゲームコンソールが進化しきれないのはテレビという最終出力による制限が大きい。テレビ以外への出力を示唆している点では評価ができるものの，テレビを無視できない以上，フルタイムフルスクリーンアンチエイリアスは必須といえるだろう。というか，Dreamcastにしても PlayStation2にしても，その部分が最初にアピールされてなければおかしいのだが。

アンチエイリアスに対応したPowerVR2と十分な容量のVRAMを積んだDreamcastでアンチエイリアスをやっているゲームはひとつもない(私は知らない)。PlayStation2のデモ映像でもほとんどない。「永瀬麗子が颯爽と歩く」とはいえ，床のテクスチャは縦解像度の不足とZ方向のエイリアスでちらついており，スポットライトを浴びても影は落ちてない。トライリニアフィルタリングすらまだ一般的ではない。

Dreamcastで採用しているPowerVRは当初からシャドウボリュームの生成機能を持つ。まだちゃんと使いこなされていないのだが，セガのシェンムーのデモ映像などを見るとオブジェクト同士での影落としをやっていることがわかる。

PCの方面では影を落とすことが普及しかけている。単に陰影がついているだけだと，オブジェクト単体ではそれなりに自然に見えても，シーンにしたときに不自然になってしまう。CGっぽさの代表みたいなものだ。

で，キーになるステンシルバッファというのは，もともとマスキングに使われるもので，描画を行う領域と行わない領域を簡単に区別するためのものだ。

それを使って物体の上に影を落とす。

一度レンダリングした画像に対し，光源から各オブジェクトエッジで作られるシャドウボリュームをステンシル内にレンダリングする。前にレンダリングしていたときのZバッファ値を残しておけば，シャドウボリュームが綺麗に物体の上に生成される。あとはこのマスクを利用して影の濃度を重ね書きすればできあがり……と，理屈を聞けばなるほどと思う巧妙さだ。最近ではエッジのくっきりした影でなく，半影生成のアルゴリズムも完成されているという。

ステンシルバッファなしでも，別画像のαプレーンで代用してやれなくはないだろうがちょっと無駄がある(パワーはあるんだろうが，VRAM容量は不安)。PCゲームではすでに頑張って半影らしきものを出しているものもある。今後は影くらいは常識として要求できる要素だと思う。

現状でPlayStation2のデモムービーをいくつか見ると影のないものがほとんどだ。あってもいわゆる「嘘影」がついてるだけ。いわゆるクリエーター側の意識が低いのでしかたない面もあるのだろうが，システム側で対処しなければレベルは上がらないだろうと思われる。

さらにいえば，このクラスの処理能力を持つからこそ求められるものもあるだろう。CG業界のトレンドはリアルタイム環境に落ちてくると書いたのだが，最近の流行はなんだろうか。

布だと思う。MAYAやLightWave3Dで布用のプラグインが出てきたので，そういったものの表現が今後は多く出てくるだろう。PlayStation2なら，リアルタイムにこなせるのではないかと思われる。少なくともデモ映像で見る限り，FF8のダンスのデモや永瀬麗子を見ても，身体にぴっちりした「モデリングされた」服しか着ていない。これでは興醒めだ(FF8のムービーを作った時期にスクウェアはまだMAYAを使ってなかったんだろうか？)。映画の画質を狙うということであれば，女の子はひらひらのドレスを着て，そよ風のなかを歩いていなければならない。

材質表現，ポリゴン数，干渉判定，流体演算……処理系の実装を考えると気が遠くなりそうだが，そのレベルの表現が要求されるハードであるということだ。どこが最初にやってくるかはわからないが，実現可能な範囲だと思っている。なぜなら業界ではすでに実際にできていることだから。

DressingSim

Fashion evolves in CG Digital Fashion Media

TOYOBO CO., LTD.

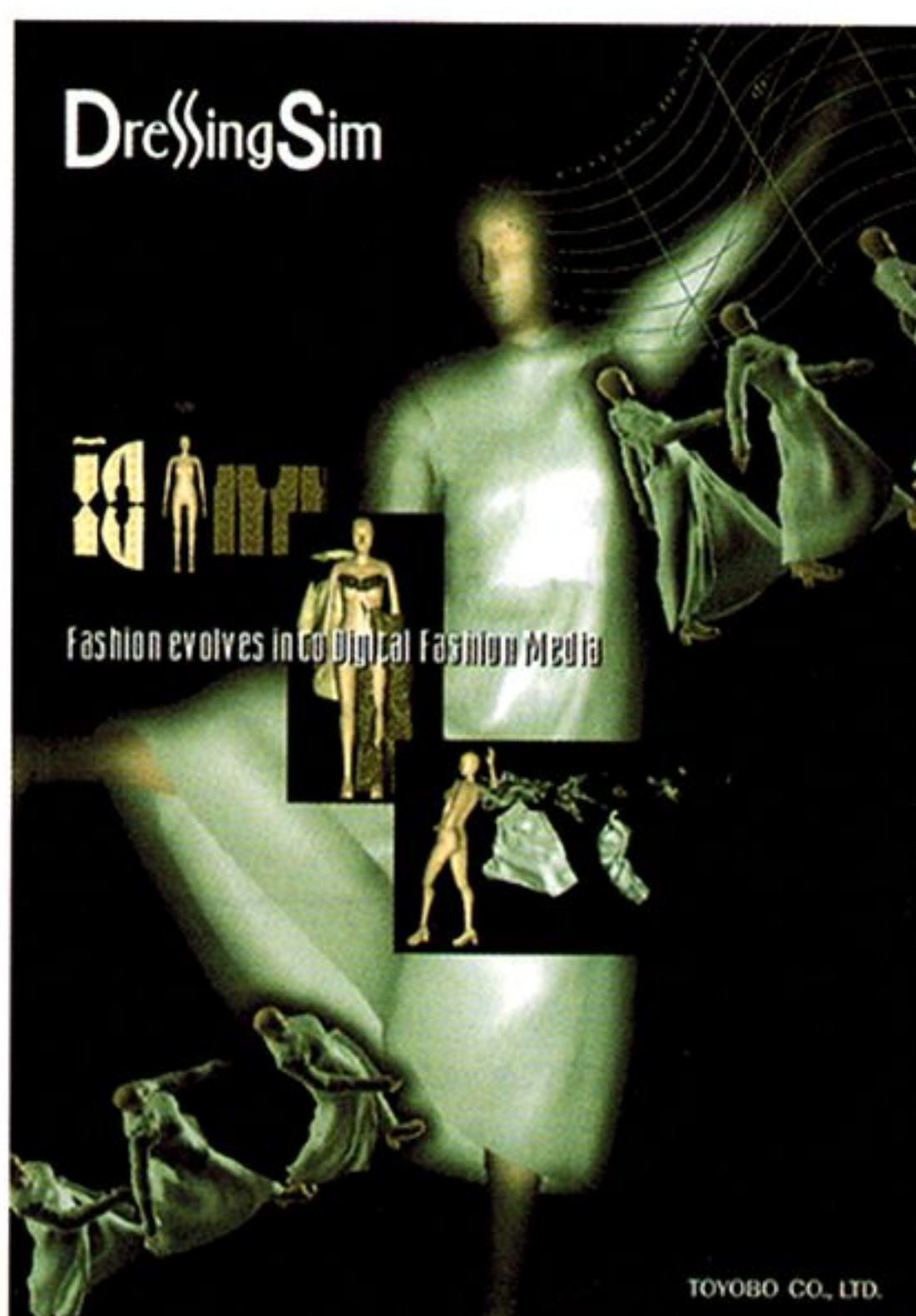



図22　東洋紡のDressingSim

シーケンス：電子ファッションショー

ドレス：ワンピース
布：ポリエステル、目付け133g/m²，摩擦係数0.5，跳ね返り係数0.5
フィギュア：身長165cm，バスト88cm，ウェスト60cm，ヒップ86cm
環境：重力1G
レンダリング：3D Studio MAX2
動作：モーションキャプチャー

DressingSimによる3Dアニメーションの製作工程

1 骨格のあるフィギュアを作成する。
2 フィギュアに着せるドレス(型紙)を選ぶ。
3 選んだ布の力学特性をインプットする。
4 フィギュアの周りに型紙を配置する。

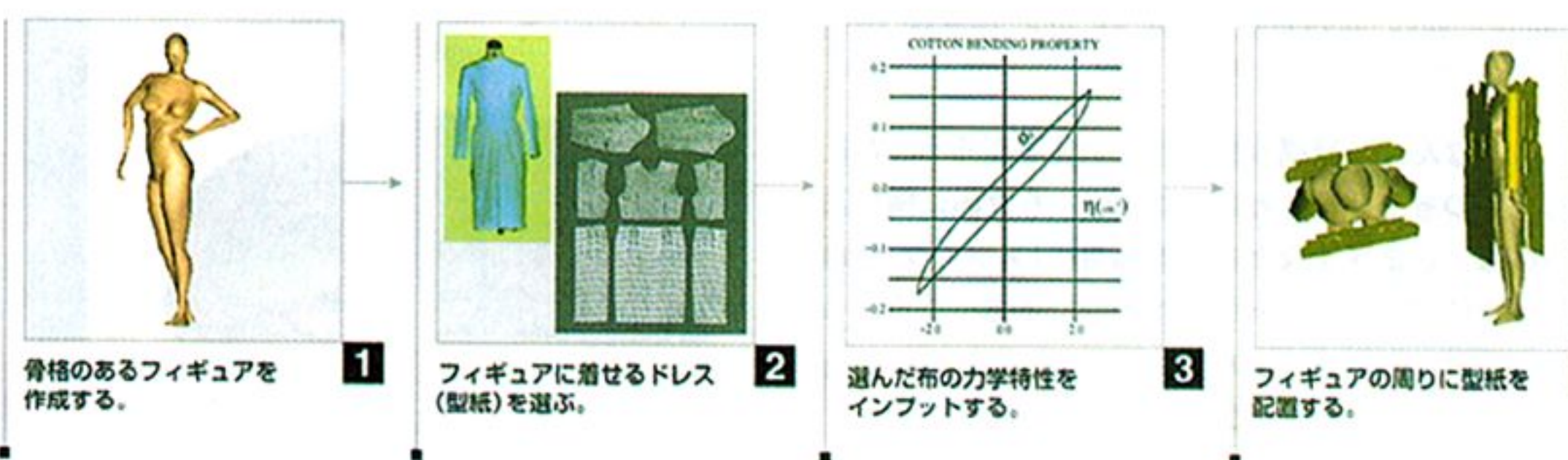



図23　布の材質などを指定してシミュレート可能

利用できるポリゴン数の増加であちこちリアルな表現ができたとしても，この辺りがついていかないと表現の幅は狭くなりすぎる。いずれにせよ，どうせクリアしなければならない壁だ。顔だけリアルな藤崎詩織など誰も求めていないと思うのだ。

PlayStation2のライバル

結局，自由度3までしかできないのなら，Dreamcastと大差ないというのが率直な印象だろう。ポリゴン数をひたすら増やすことは簡単にできるのだが，意味があることなのかどうかはわからない。もちろん，余裕があるのは悪いことではないが。

なにやらPlayStation2に自信を持って，ソニーグループ一丸となってマイクロソフトに喧嘩を売っているようにも見えるが，PCってのはやはり侮れない。現状で3Dゲームの最先端はやはりPCで展開されているように思われる。

300MHz駆動の9個の浮動小数点積和器。これがPlayStation2の生命線だ。一方でPentium Ⅲを見てみよう。500MHz駆動の4個の浮動小数点積和器を持っている。スループットは2クロックとあまりよくないみたいだが，本体FPUがスループット1の2並列構造なので，合計すれば演算能力は侮れない。除算器などの存在を除外すれば，PlayStation2が圧倒的に大きなアドバンテージを持っているわけでもなさそうなのがわかる。そして，PlayStation2の登場するであろう時期にはPentium Ⅲは1GHzの領域へと突入していく。コストの問題などもあるが，重要なのはさらに半年もたてば，それがPCでは当たり前のプロセッサになるということだ。もちろんハイエンドはもっと先に進んでいく。PCの進化は速いのだ。

描画関係も同様だ。私は昨年，当時ほぼハイエンドにあたるVoodoo Bansheeのビデオカードを買ったのだが，近々登場のVoodoo3はBansheeの2～4倍の描画能力を持つと聞いてあきれている。最強の座にあったRIVA TNTのマイナーチェンジ版がTNT2だ。基本部分は無変更のままプロセスルールを変えてクロックを上げるだけということで，たいしたことはないだろうと思っていたのだが，100MHz未満で動いていたものが160MHz超で動くと，さすがに性能も跳ね上がってくる。しかし，nVIDIAの本命は年末の「大物」だ。

かつて久多良木氏は「PCの進化はちまちまとして近視眼的だ。うちはどーんと世代を超えていく」という意味のことをいってたのだが，技術革新で大きくリードしようとするSCEと小刻みで着実に進化を続けるPCは，ウサギとカメのレースに似ているようにも思えるのだ。

まあ，確かに終わってみれば「ビルゲイツの勝利」では世の中面白くないのだが。

家電とのリンク辺りはまったく未知数。ソニー本社がしゃしゃり出て主導権を取ってしまったので，状況はさらに混沌としているように思われる。おそらくまだまだ隠し球はあるのだろうし，実際の製品なりデモ機なりができてからまたレポートすることになるだろう。

NINTENDO2000とArtX

「え，まだなんかやってたの？」
という感じの人も多いと思うが，任天堂の次世代機の話が急浮上している。前々から噂がなかったわけではないのだが，PS2が発表されたあとでも消えていないというのがなかなか凄い。もともとPS2の発表に対して「いまもっと凄いのを作ってる」発言をしていたのだが，

現在のところ判明しているのは「MIPSアーキテクチャではない」けど「下位互換性がある」のだが「ROMスロットは持たない」という謎のような事項だけだ。128ビット2CPU構成，という噂が出回っている。バスは400MHz。NINTENDO64を現状の技術で作り直せば結構イカしたマシンになるというのは，以前から

しかし，R4000をエミュレートできそうなパワフルなCPUというと，さほど多くはない。一般的なものだと，せいぜいAlphaかPowerPCか，PentiumIIIとか……はないだろうなあ。さらに現状では128ビットCPUと呼べるチップはEmotionEngineしか存在しない。

予想される性能は「PS2との差が30%以内」で20人もの旧SGIスタッフがスピンアウトして作った"ArtX"テクノロジーが投入されるという。

実際にどんなものになるのかはまだまだ不明だが，この「噂」はそれなりに信用できそうなソースから出た話だ。「PS2クラスのマシンですら，もはや飛び抜けた存在にはなりえないのではないか」また「下手をするとN2Kが先に出てくるんではないか？」といった予測もされているようだ。詳細がわかるのは……秋以降なのかなあ。

column

インターネットの活用？

Webの世界ではゲーム業界に直接関係はなくても，噂レベルであればいくらでも情報が入ってくるのだけど精度の高い話はなかなか入ってくるものではない。新製品関係では誤情報も多い。それらのものをひっくるめて，はっきりいってしまえば，たいていの情報はWeb上に落ちている。デマも多いので注意が必要だが。例をひとつ。TAMAR'S HOME PAGE（http://www.unc.edu/~tamar/）のゲームニュースアーカイブから引用しよう。

```
Aug. 27, 1998
Rumor: Sony has showed a preliminary mainboard of the the PSX2 to select developers. It supposedly consists of a main and sub processor, with the main processor being a new 128bit chip and the sub processor is the R3000, the original PSX processor. Both the main and sub processor can run independently, so it seems like PS2 could be backward compatible with the current PSX. Codenamed the T-Rex, it is planned to have a modem and audio/video input. It's still unknown if the system will be CD, DVD, or a hybrid like the Dreamcast.
```

これなんかは結構な精度だといえるだろう。ただし，いつもこういくわけではない。ちなみに横にあるのは，さまざまなところで独自に「スクープ」された次世代PlayStationのデザインである。こういうのは海外の情報サイトを見て回っていればいろいろ見つかる。コメントもなかなか面白い。

「おい，電源ボタンはどこだ？ コントローラ側かい？」

ソニーと東芝がチップを発表するという噂が出ると「SCEがplaystation2.comとplaystation2000.comのドメイン名を買ったらしい」とかいう情報も一緒について流れてくる。たまにチェックするくらいがいちばん楽しめる感じだ。

インターネット上にはたいていの情報は転がっている。ただ，ちゃんとした情報を見分けることのできる人は，たいていすでにその情報を必要としていないというのがジレンマなんだろうなあ。

川原由唯
都築和彦
末次徹朗
森川久志
田中順子
由水桂

Visual Laboratory

「絵を描く」ないし「映像を作る」のに決まった方法はない。それぞれの作品に対してどのような方法をとろうと自由だ。
また，評価法にしても，上手い下手といった簡単な尺度では測れないものもある。
解像度や色数の制限がほとんどなくなった現在では，
技法そのものよりも，表現しようとする内容のほうを重視すべきだろう。
CGはより自己表現のできるメディアとして完成されつつある。
とはいえ，かつての制限された表現範囲内で
表現力を上げる技法などがまったく無駄なものだったのかというと，そうではないだろう。
どのようにすれば絵を整えられるのか，
そういったことに対する理解があるかどうかでもできあがる絵は変わってくるように思われる。
ドット打ちの経験のある人は，おそらくそこから学んだことも少なくないはずだ。
CGにせよなんにせよ，「作品」として提示されるときには「結果」しか与えられない。
なにかを作ろうというときには，その結果だけではなく，過程がより多くの意味を含んでおり参考になる部分も多い。
ここでは，作品制作の過程を通した作品作りを見ていこう。

川原由唯
Youi Kawahara

お魚を描く

**今回は魚を主人公にしてみました（本当か？）。
テクスチャ貼り付けのレシピっちゅーことで，どうかおつきあいのほどを。**

今回もPhotoshop ver.4.0オンリーです。ver.5.0はインストールはしてるけど，重くてサイテーなので使ってません（そういう人も多いみたい）。いまだに4.0ばかり使ってます。

Photoshopはどんなにメモリを積んでも必ずディスク上にスワップを作るらしいので，「レスポンスを速くしたいならRAMディスクにしたほうがいいよ」とMacintoshを買ったときにWoody-Rinn氏に教わったので，現在では搭載メモリの約半分をRAMディスクに割り当てて，Photoshopのスワップに指定しています。今回は変形系のフィルタをかけるときにメモリ不足が発生して苦労しました（でっかい画像のまま処理しようとするのがいけないんだけどね）。やはりメモリは湯水のように必要。いくら積んでも満足することがありません。

お魚を描く

シーラカンスのような魚の下描きです。一応，図鑑などを見ながら描いてますが，古代魚みたいになればそれでよかったので，多分シーラカンスにはなっていません。例によってケント紙の鉛筆画をスキャナで読み込んで，コントラストの調整をして，取り込み時のゴミを飛ばしています。

下書きの線は，絵の具を載せる前のセル画の線画のような状態に加工しています。レイヤーオプションの「乗算」でも問題ない場合が多いんですが，線画に変換しておいたほうがPhotoshopの処理が軽くなるのと，「乗算」ではどうしても対応

制作環境
本体：Power Macintosh 7500/100
アクセラレータ：MACh SPEED 604e 233MHz
搭載メモリ：352Mバイト
VRAM：4Mバイトに増設
ハードディスク：内蔵1Gバイト＋4Gバイト
ディスプレイ：17インチトリニトロン管
タブレット
フラットベッドスキャナ：JX-250
その他，230MバイトのMO，フィルムスキャナなど

使用ソフト　Photoshop 4.0

できない処理があるためです。

セル画状レイヤーへの加工の方法はいろいろあるようですが，ボクは次のようにしています。

- 原画を白い背景，黒い線で読み込んでおきます。画像のモードをRGBにしておきます。
- 原画の上に，セル画にするための新規レイヤーを作成します。
- チャンネルパレットで，「赤」のチャンネルを選択しておきます（別に緑でも青でもいいんですが，便宜上赤にしておきます）。
- 点線の丸が描かれたアイコンをクリックしてチャンネルを選択範囲として読み込みます。
- 「赤」チャンネルから「RGB」チャンネルを選択し直して，赤青緑すべてのチャンネルを操作できるように戻しておきます。
- レイヤーパレットで，先に作成しておいた新規レイヤーを選択します。
- 「選択範囲」の「選択範囲を反転」を実行します。
- 「塗りつぶし」で黒で塗りつぶします。
- 選択範囲を解除します。

以上でセル画レイヤーの完成。上記の方法はタカベ氏に教えてもらったものですが，ほかにもいろんなやり方があるみたいなので研究してみてください。線の色は色調整で変更すれば，セルアニメの色トレースのようなことも可能です。ボクは茶系の線が柔らかくて好きなので，よく茶色に変更しています。

図1　ケント紙に描いた下絵をスキャナで取り込みます。この段階ではゴミが残っています

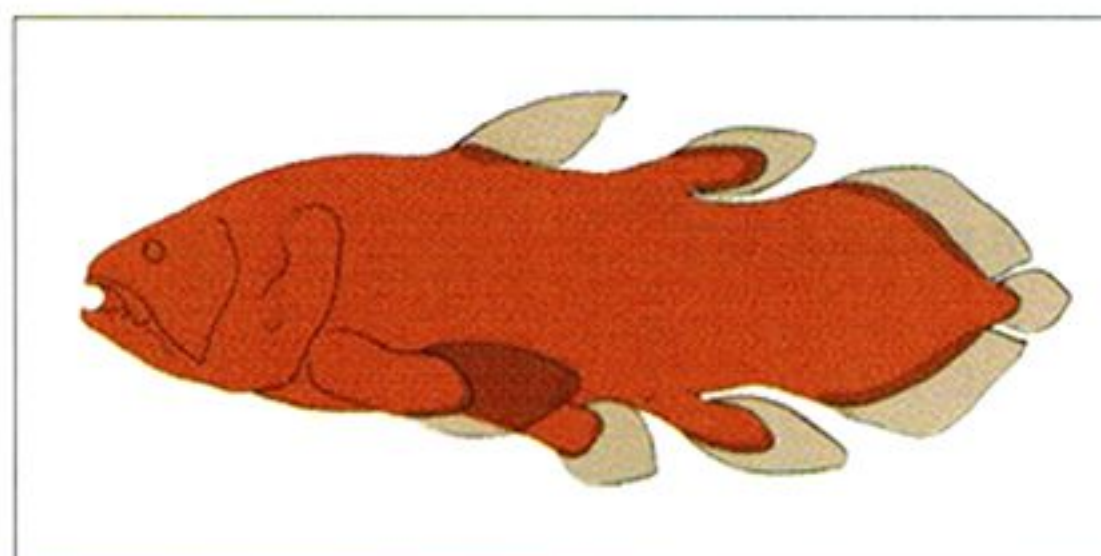



図2　下書きにあわせてべた塗りしたもの。別レイヤーにしたヒレの部分は濃度を落として薄さを表現します

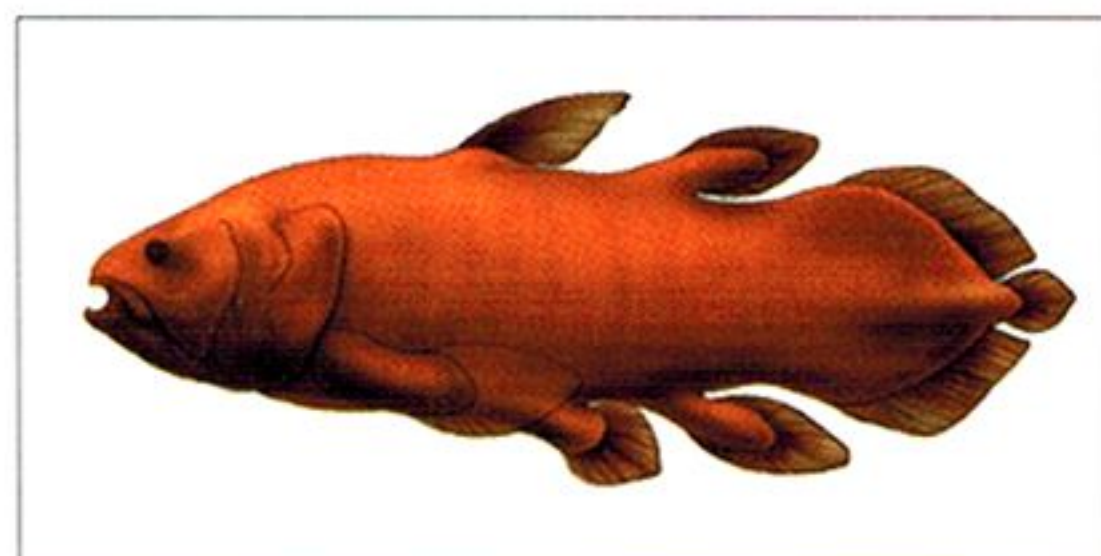



図3　「覆い焼きツール」や「焼き込みツール」でタッチを入れて立体感をつけます。どうせあとでウロコを載せてしまうので，あまり丁寧には仕上げていません。ヒレも同様に陰影をつけて，「指先ツール」で繊維の流れを出します

ウロコテクスチャを作成する

図4　ウロコのテクスチャを作成します。透明レイヤーの上に円を描きます。これを1枚のウロコにします

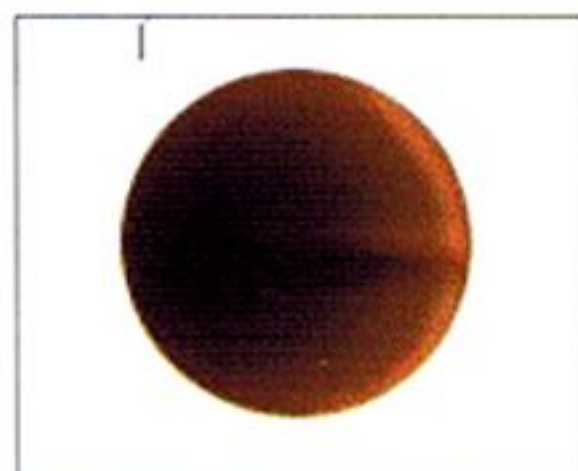



図5　またまた「焼き込みツール」や「覆い焼きツール」を使って立体感をつけます

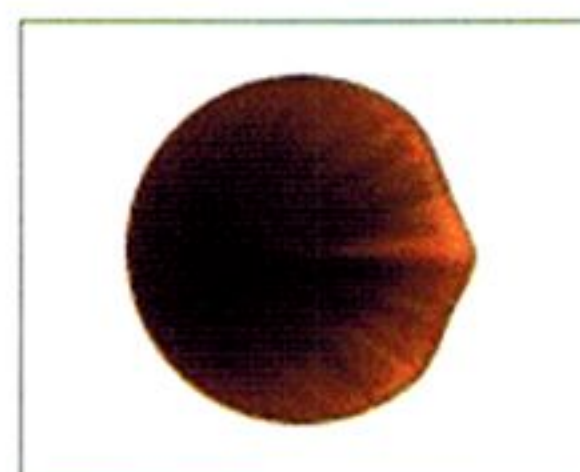



図6　ただの円形だとさすがに手抜きなので，「消しゴムツール」で円の縁を削って形を整えます。「指先ツール」で放射状のタッチを入れます。本来ウロコは年輪状に成長していくものらしいですが，今回はそこまでこだわってはいません。これでウロコ1枚完成

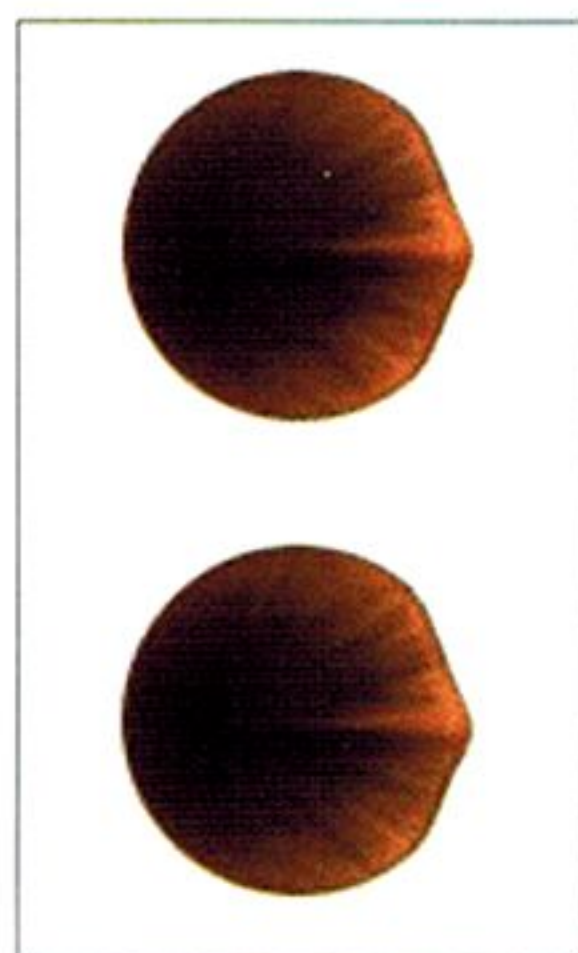



図7　完成したウロコのレイヤーを複製し，位置をずらします

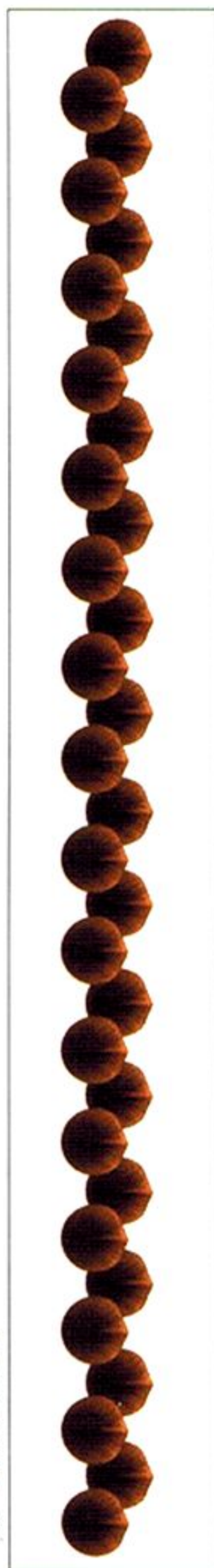



図8　同様にレイヤー複製と位置移動を繰り返し，瓦状に重ねていきます。ウロコの重なり方は，魚の頭の側にくるもののほうが，尻尾のほうよりも上（手前）にくることに注意（当たり前なことだけど，最初なにも考えず作業してたら逆に重なっていてあわてて修正した）

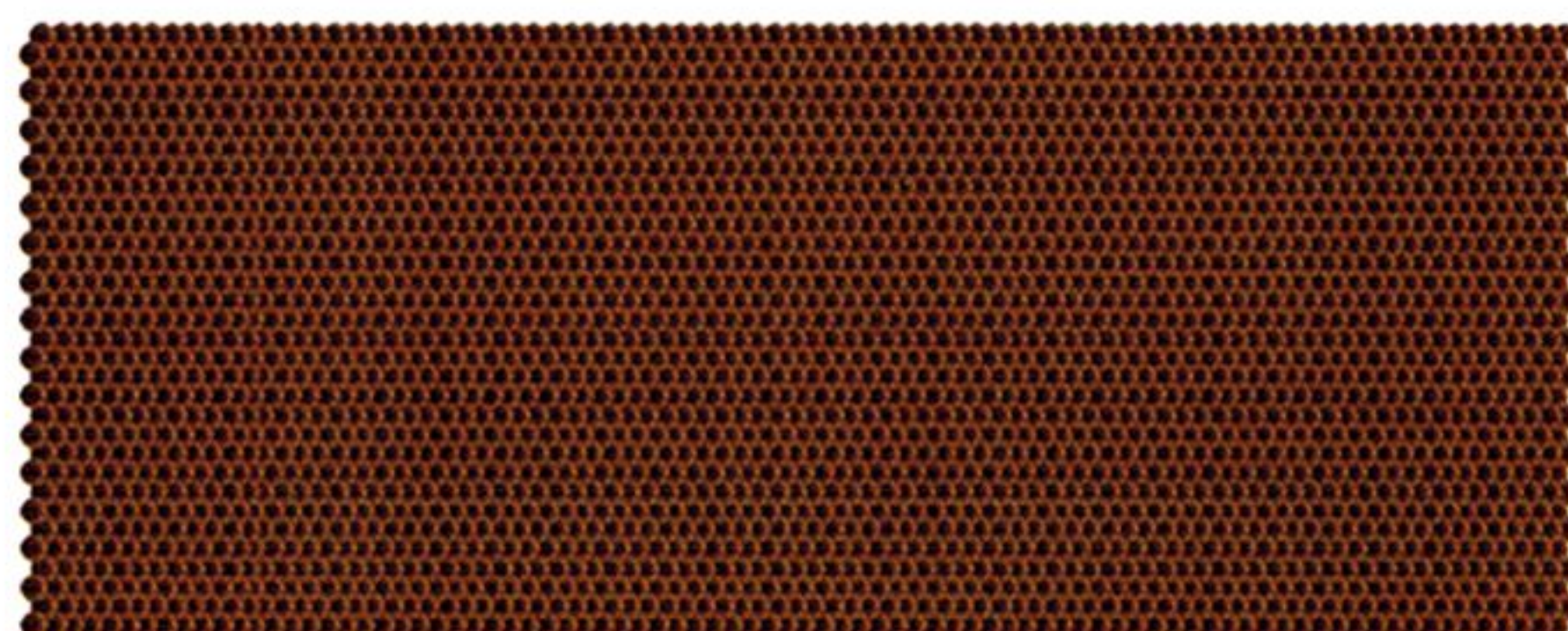



図9　適当な大きさになるまで増殖させ，1枚のウロコテクスチャ画像にします。魚の種類でウロコの密度や大きさは違うので，こだわるときにはこだわれるところ

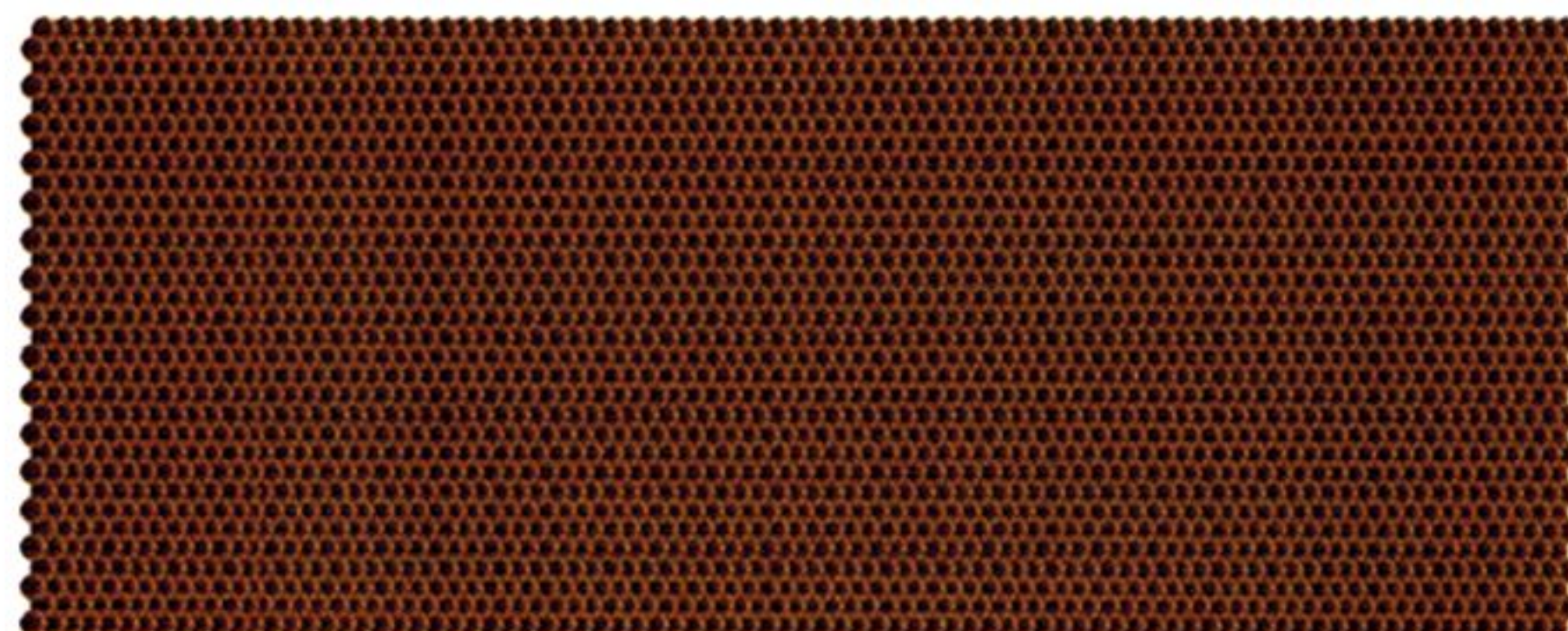



図10　スクリーントーンよろしくこのまま魚に貼り付けてもいいのですが，せっかくCGなんですから，フィルタを使って魚体にあわせた変形を行います。形状変形「球面」の円筒モードで，横向きの円柱にマップするように変形します

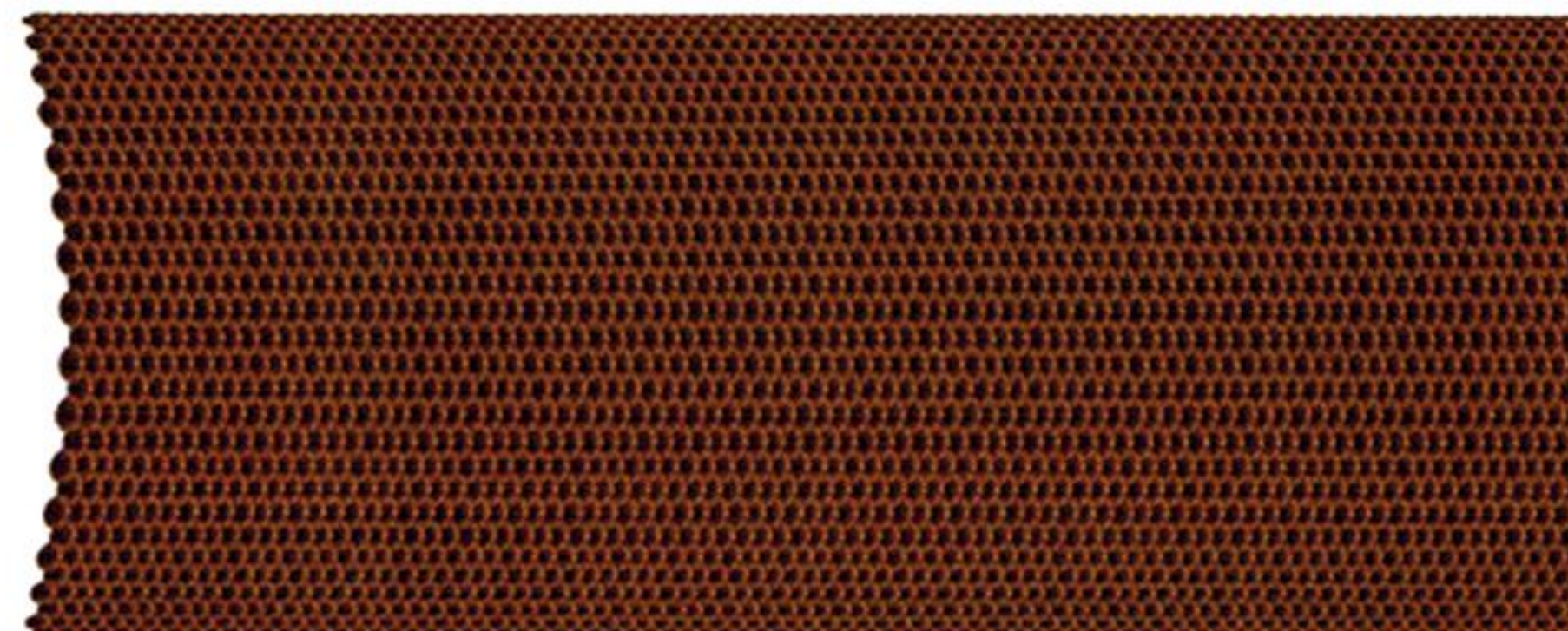



図11　形状変形の「シアー」を用いて中心軸の方向を若干尾部にずらします

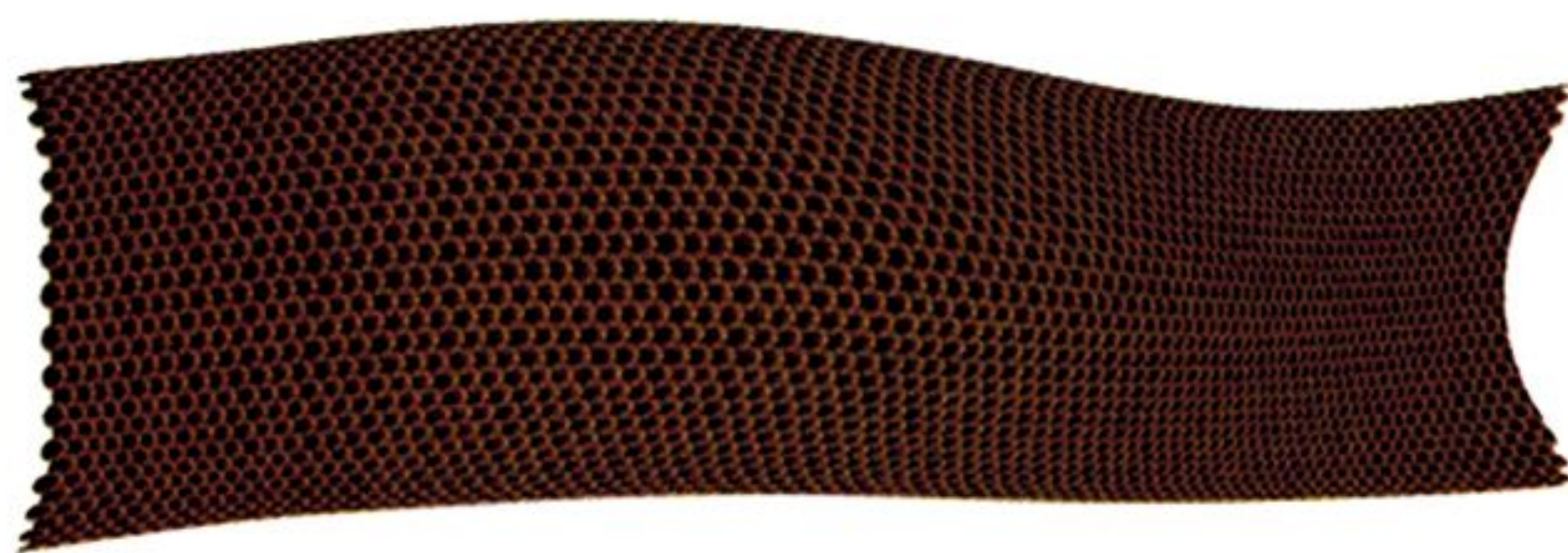



図12　形状変形の「つまむ」フィルタを用いて，魚の腹のふくらみを表現します。逆に尾の付け根のあたりはマイナスのパラメータで絞ったように見せます。「つまむ」フィルタは，画面の真ん中を基準につまんでくれるので，つまみたい部分が画面の中心になるようにレイヤーを移動してから実行します。自由変形を使って尾部を小さく縮めています。

背骨が上下に波打つ感じにあわせて「シアー」で歪めます。これでウロコテクスチャが完成。あとは魚体にあわせてカットして貼り付けるだけ。下書きを重ねあわせて変形しているわけではなく，目分量でかなりいい加減に加工してるんですが，テクスチャとして貼り付けてみるとそれなりに格好はつきます。この辺の処理をするのとしないのとでは，完成時の存在感が全然違います

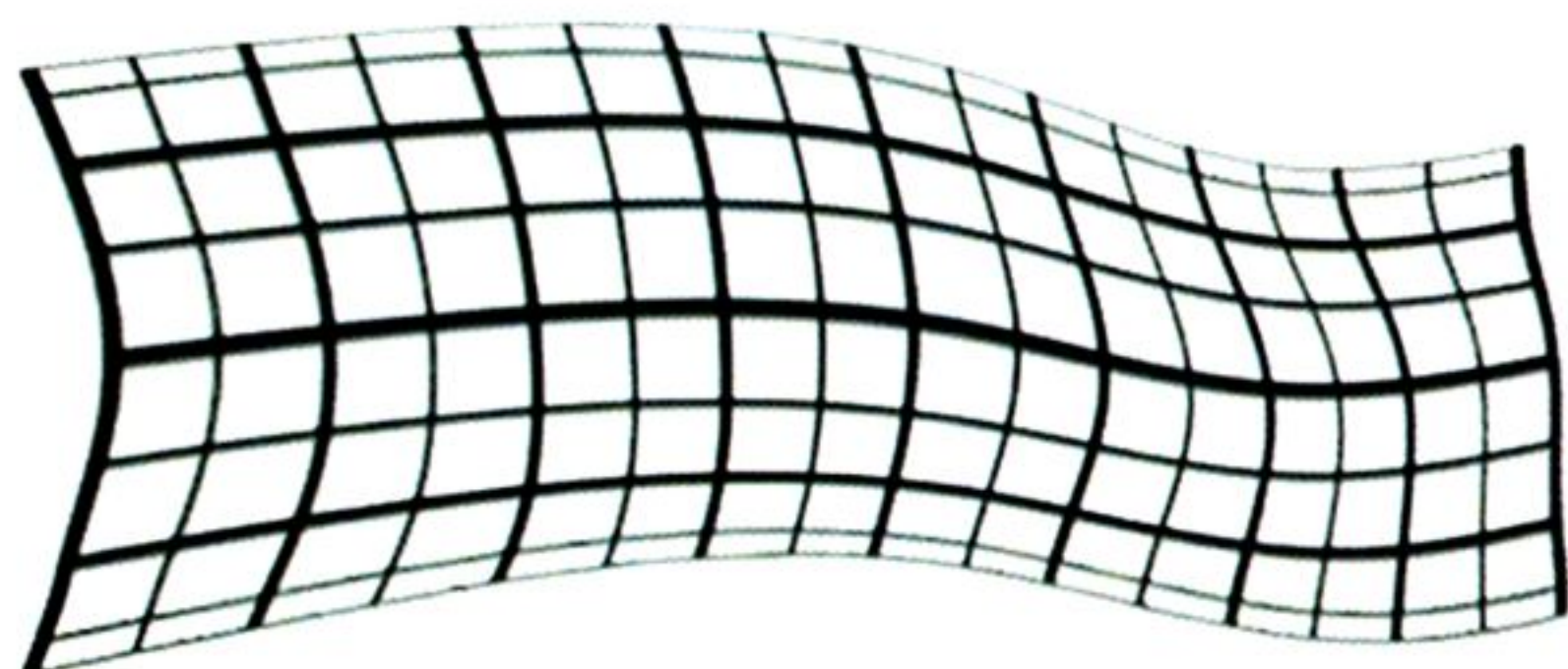



図13　変形の目安はこんな感じ。あまり正確ではないですが

魚にウロコを貼り付ける

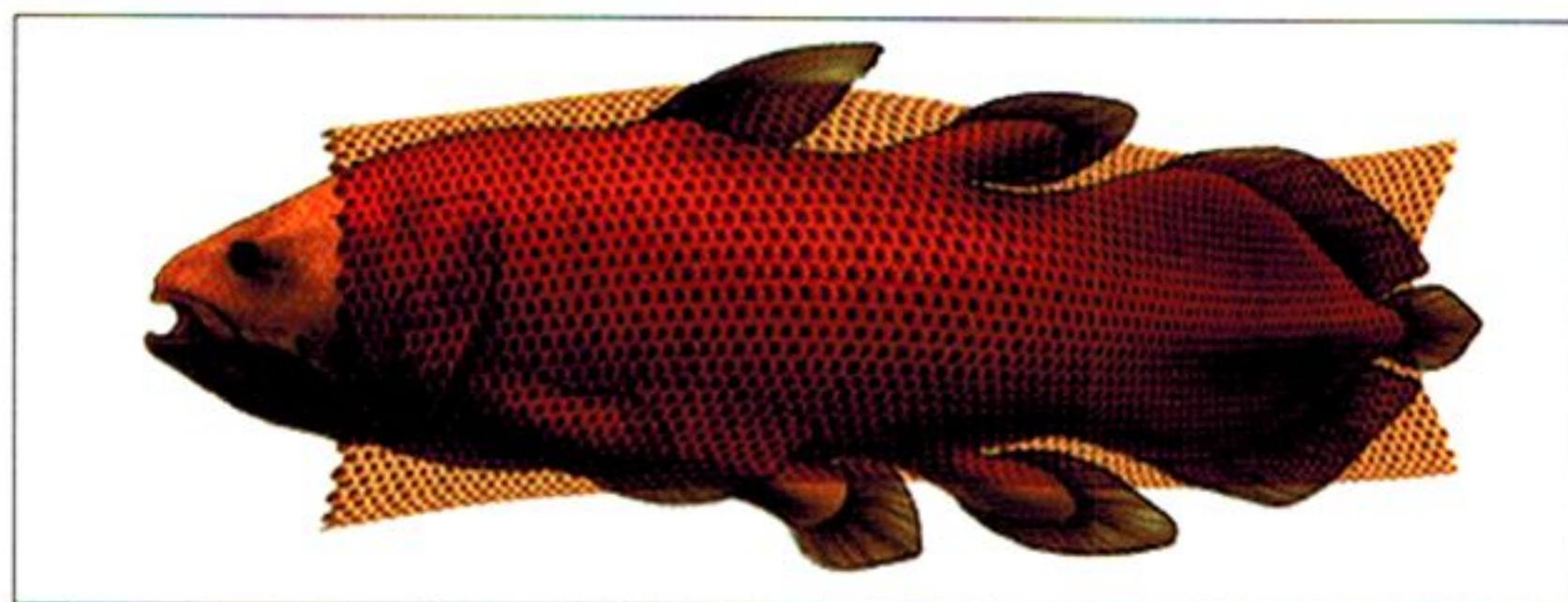



図14　完成したウロコテクスチャを魚の上に載せて位置あわせをします

図15　魚の形にあわせてテクスチャをカット。頭の部分など，ウロコの途切れる境目は，ぼかした消しゴムツールを使ってならします

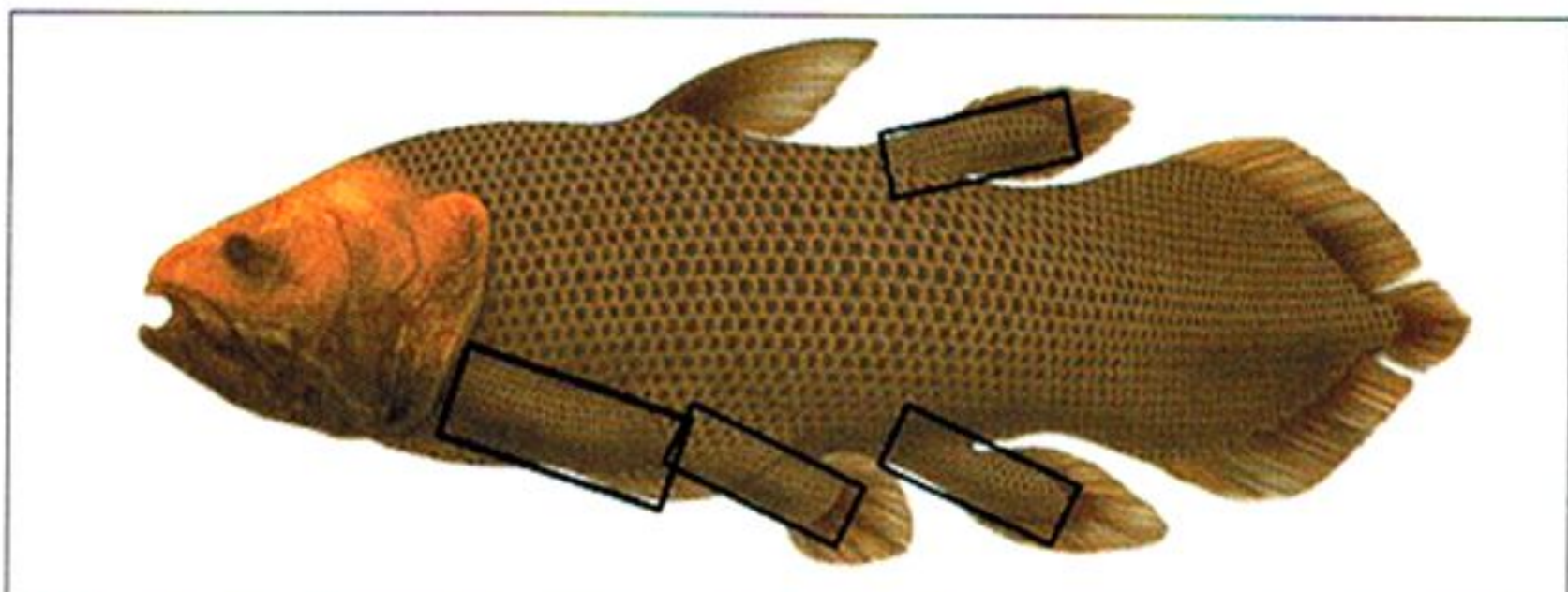



図16　ヒレの付け根の部分も，同じウロコデータを使ってマッピングしてあります。形状変形の各種フィルタを使うのは胴体部分と同様。胴体との接合部分は，やはり消しゴムツールでぼかしてならします。そんなに丁寧にしなくても，なんとなく見られるようになります

図17　コントラストを調整し，大理石などの素材集から持ってきたパターンを上に載せて模様を作り，リアル感を出します。眼を描き込んで命を吹き込んであげませう(シーラカンスは眼が濁ってたほうが「らしい」かも)

図18　「覆い焼き」や「焼き込み」で立体感を整えて，魚が完成。ちょっとリアルになりすぎたかしらん

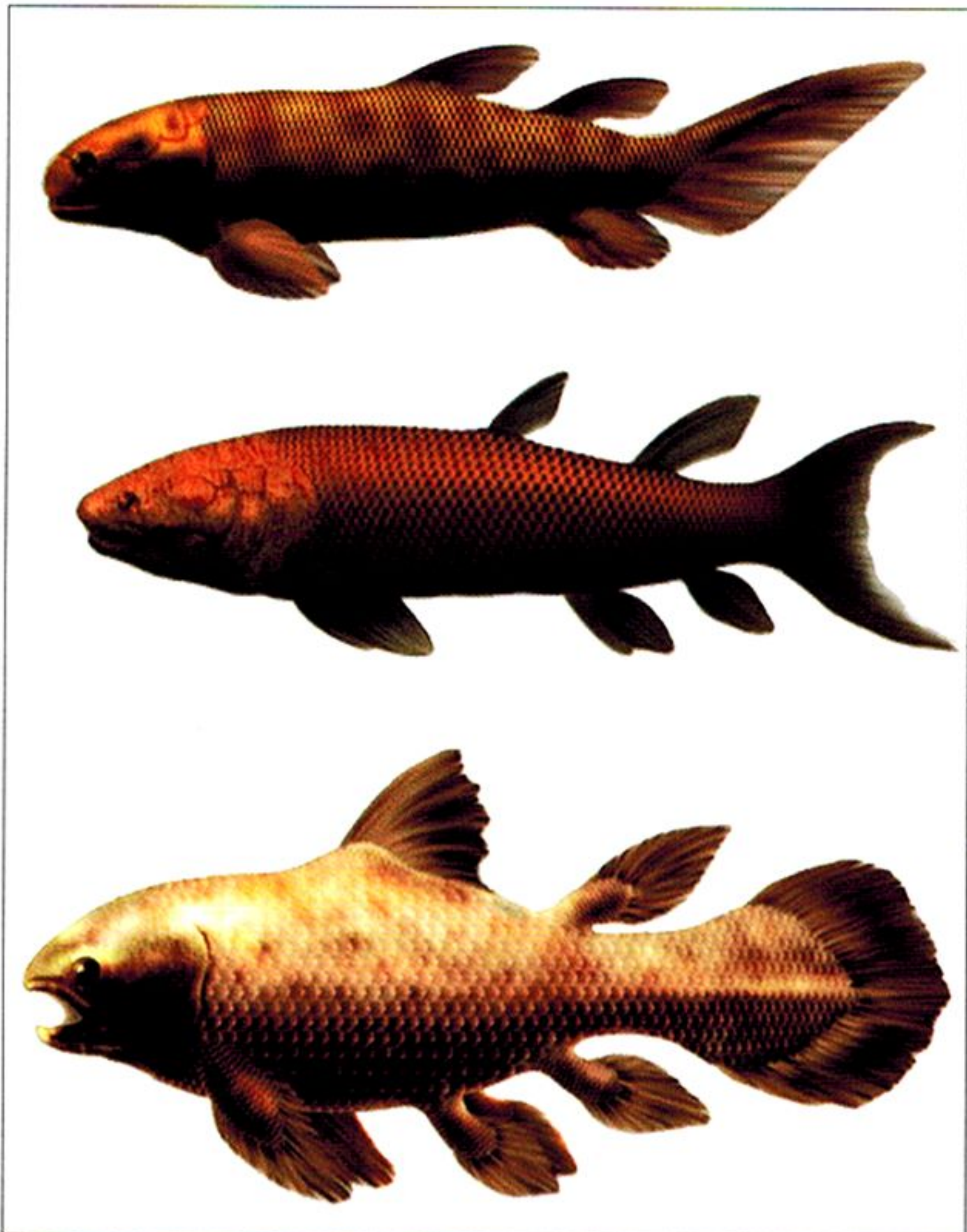



図19　同様に作成した古代魚3態。やっぱり雰囲気だけもらって，自己流にアレンジしてます。実在の魚ではありません

人物を描く

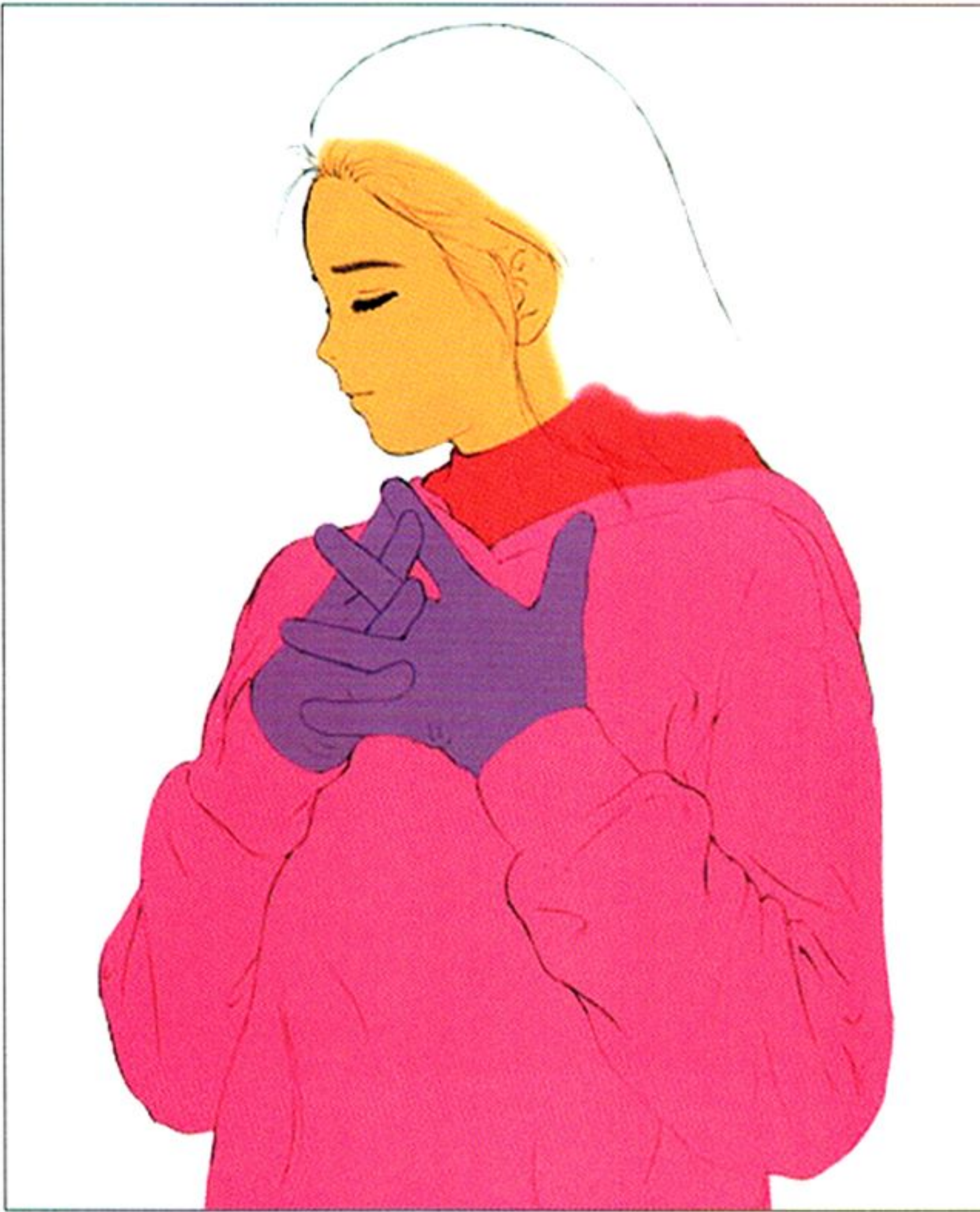



図20

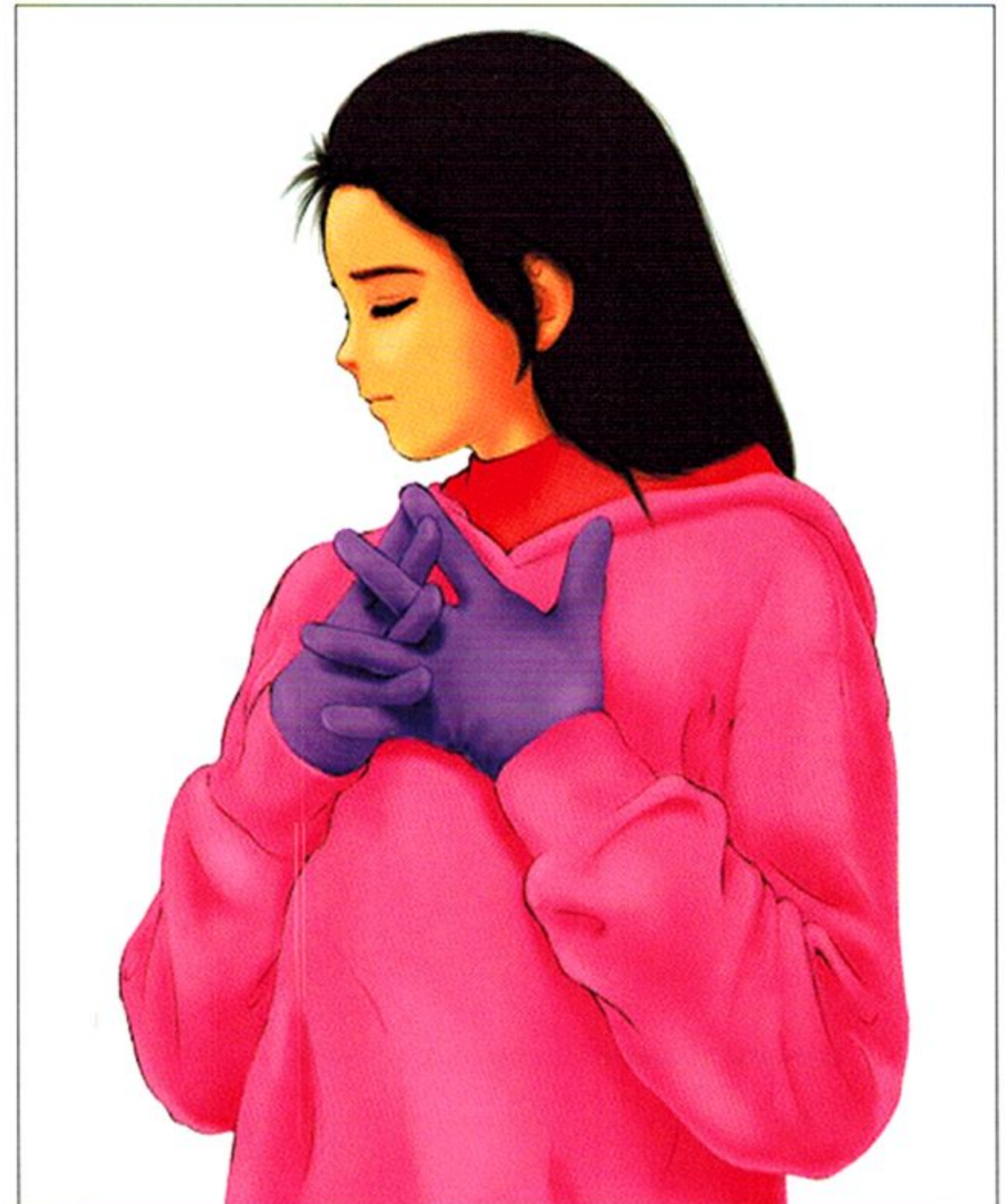



図21

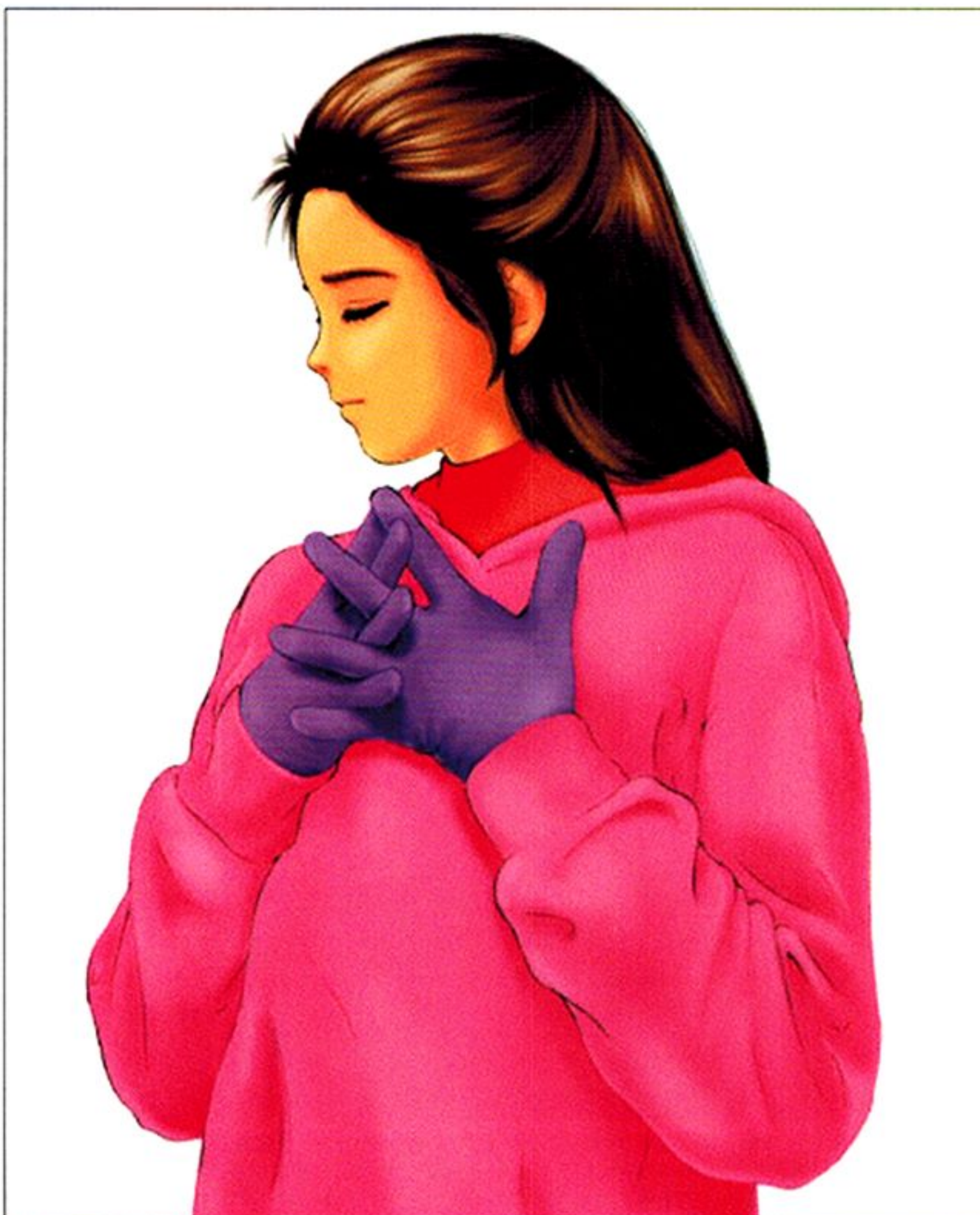



図22

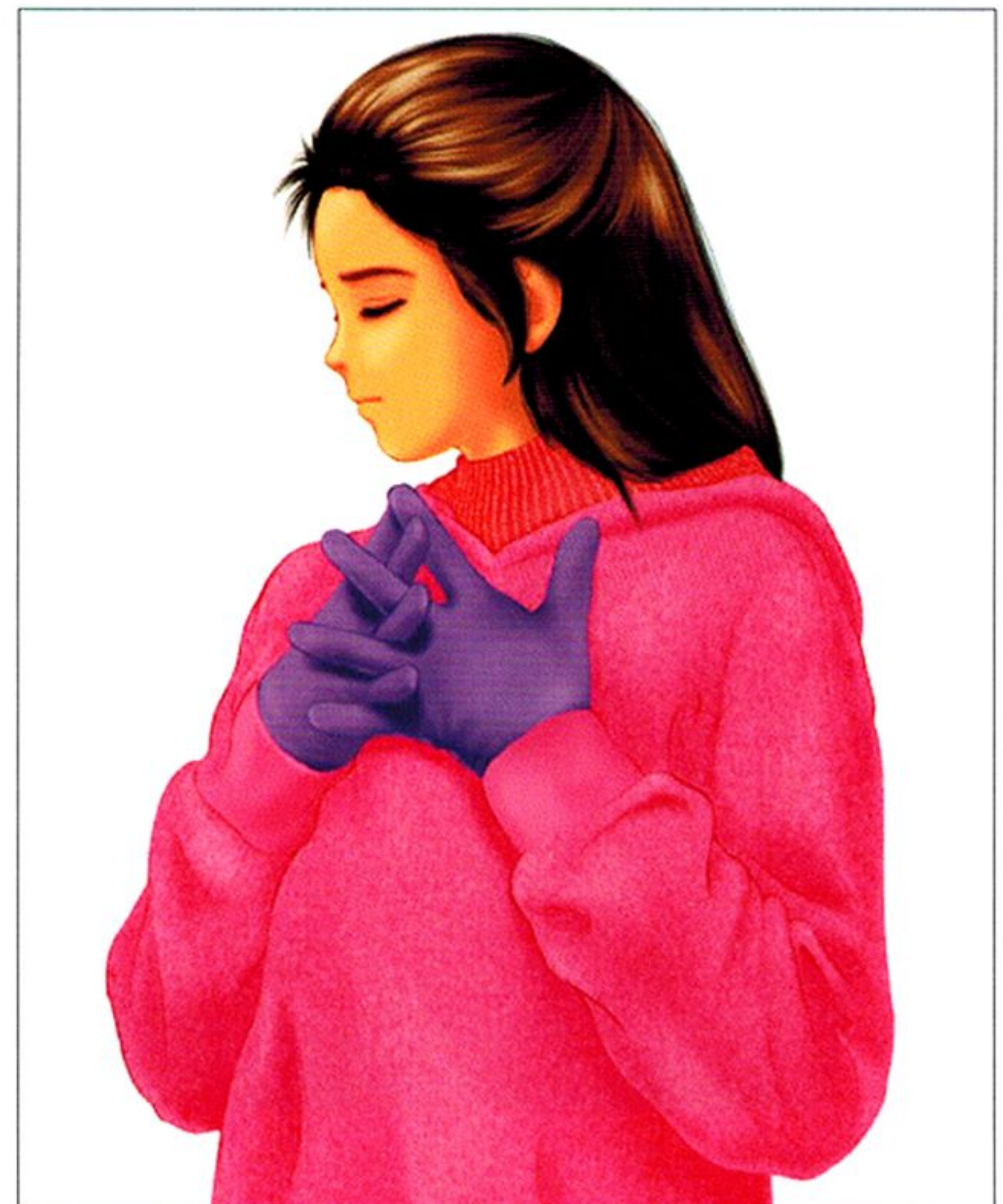



図23　少女の描き方は前号で説明したのと同じなので省略

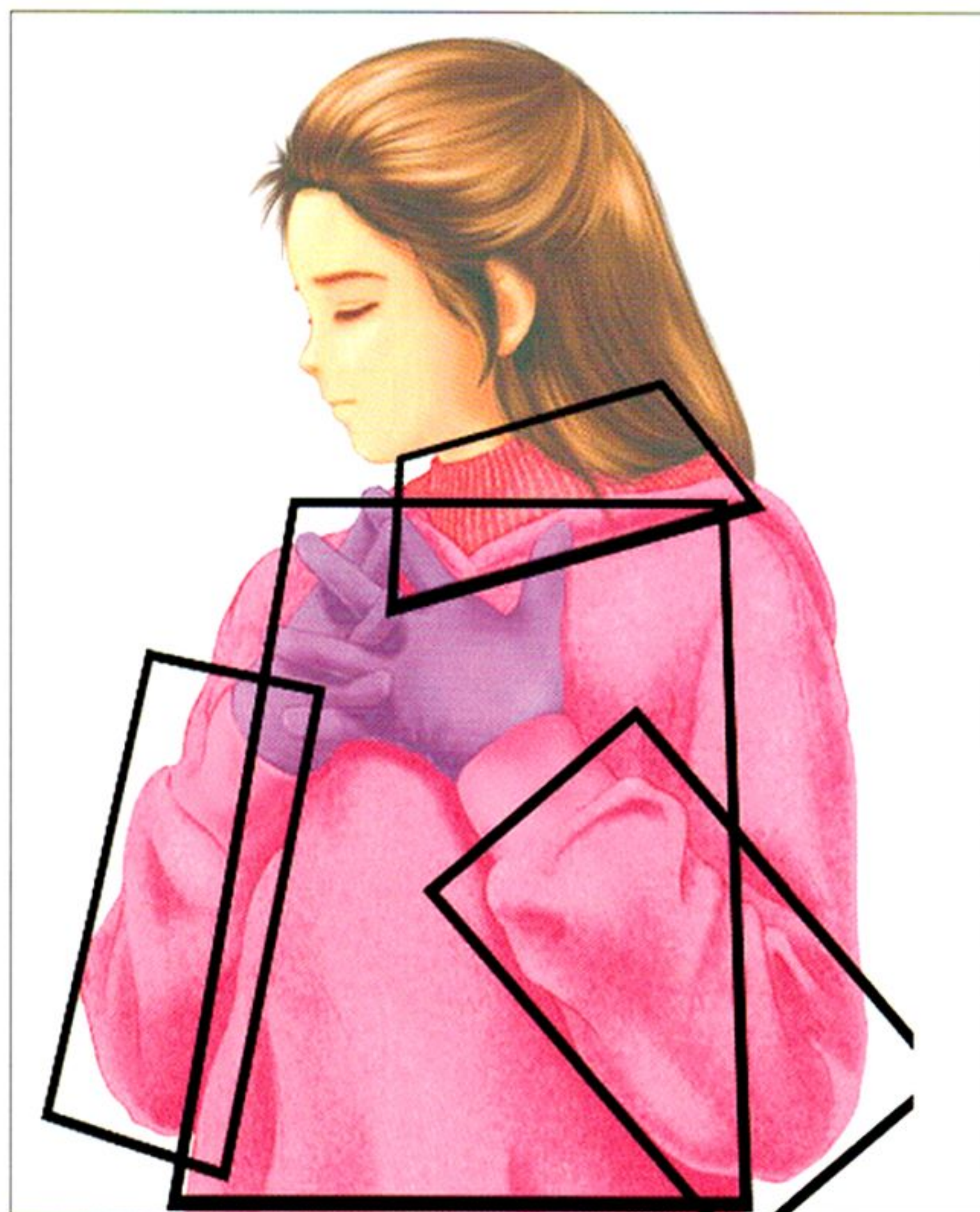



図24　少女の服のテクスチャも，ウロコと同様に変形フィルタを使って整えてから貼り付けています(が，この絵ではほとんど見えませんね)。もとになるテクスチャは，素材集などは使わず「ノイズフィルタ」や「波形」フィルタなどを使って作りました。デフォルトのフィルタを組み合わせるだけでも結構いろんなテクスチャが作れるのがPhotoshopの便利なところ

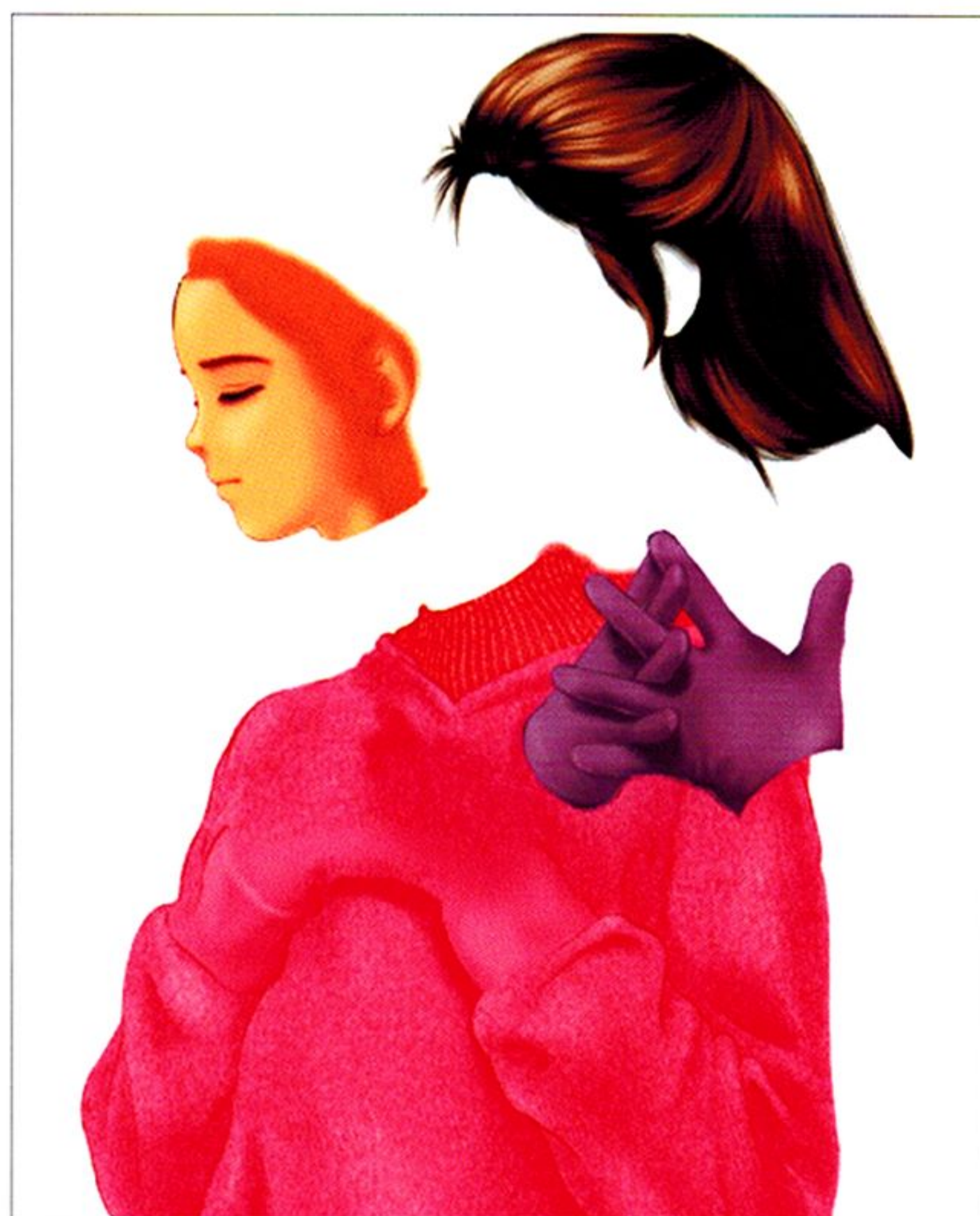



図25　サービス(?)で，レイヤーをずらしてみたところ

猫を描く

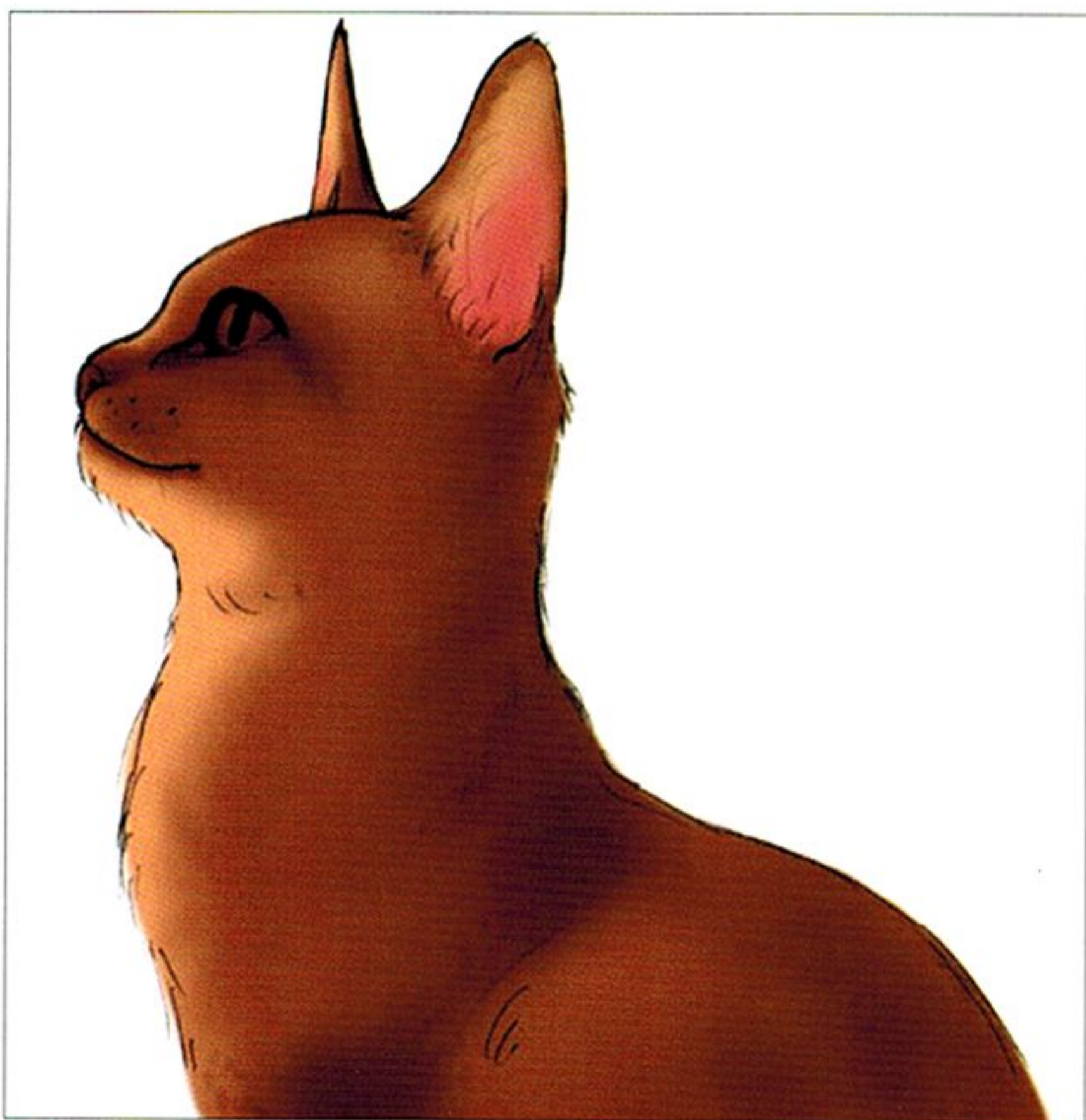



図26

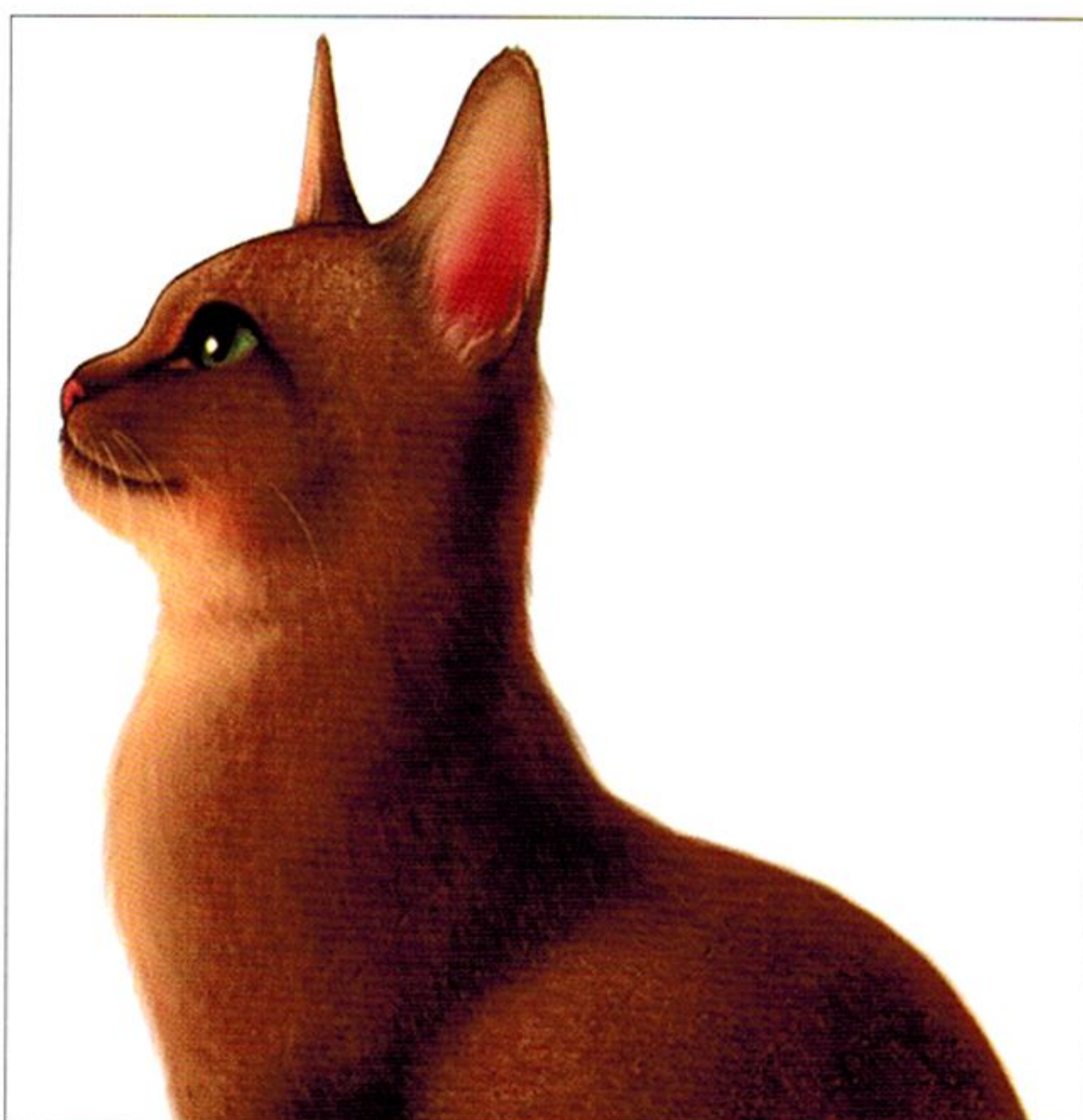



図27　猫の描き方も人物とほとんど同様で，たいしたことしてないので省略。短い毛の感じを出したかったので，「ノイズフィルタ」で作ったテクスチャを軽くボカして重ねたあとで，「指先ツール」で毛の流れの方向に軽くなぞって毛並みを表現しました。仕上がりはイマイチでしたけど，遠目に見ればそれっぽいからいいですよね

仕上げ

図28　作成した絵を配置して仕上げます。古代魚の雰囲気を盛り上げるために堆積岩っぽい岩の素材の上に魚の画像を載せて合成します。合成のモードを「比較(暗)」にすると，適度に岩にとけ込んで化石の雰囲気になります。
猫と少女を違和感のない大きさに拡大/縮小して配置します。ポートレートっぽい逆光の表現が好きなので，猫と少女の輪郭を「覆い焼きツール」で光らせています。

発表のために，画像の大きさを調整して完成。一所懸命に描いたわりには魚が目立たないですが，まぁ絵を描くときにはよくあることです(よねぇ？)。

リアルの価値

妙にリアルっぽくなりすぎた魚を見ていて思ったことがあったので，そのことについてちょっと。

最近は絵の感想もらうことも皆無になってしまって寂しい限りなんですが，以前パソコン通信がもっと活発だったころは，よくボクの絵は，CGイラスト関係のフォーラムでは「リアルっぽい」といわれてました(絵画やスーパーリアルなイラストを描いている方から見たら笑っちゃうような漫画絵だと思いますが)。実際，そういわれることについては悪い気はしないですし，嬉しくもあるのですが，実はリアルって「上手」と同義にはなりえても決して誉め言葉じゃないんですよね。ボクはそういう認識でいます。

ボク自身は絵に対して写実主義を志向していたわけでもないし，むしろ好きなイラストレータは漫画絵系の作家のほうが多かったりするんですけど，自分の能力で描ける絵を描いていたら自然にこういう画風になってしまったという感じ。写実的頭身を意識して描いていることは事実ですが，よく見ていただければわかるとおり，顔の構造は明らかに漫画的ですし，塗りや影もアニメや漫画のデフォルメ文法から派生したものになっています。実は漫画絵って，すごく好きなんです。

今回みたいなタッチのときのボクの絵は，気に入った風景や人物の写真(自分で撮ったり，雑誌から切り抜いておいたり)を座右に置いて，それを資料にして，アレンジするのに近いやり方で絵を構築していきます。もちろん写真そのままには描きませんし，自分の意匠を含めていますが，たとえ誉められてもリアルだといわれても，それは自分から生まれ出でた創造力ではないという後ろめたさのようなものがあるんですね，きっと(だから，資料もないのにサラサラと漫画絵が描ける人にはコンプレックスがあります)。

リアルに描けることと，リアルにしか描けないことが一般には混同されているようです。いや，どうやら描き手のほうでも勘違いされている方が多いんですが。

リアルな絵自体の価値や精密で写実的な描写が要求されるケースがあることは認めますが，やっぱりそれは「コピーの価値」(っていうと語弊があるかな)でしかないと思うんですよね。安易な「リアル」は結局，作家が自身の財産として咀嚼，消化できていないだけのような気がします。同じエネルギーを使うのなら，その分，感性に訴える，魅力のあるものを創造したいじゃないですか。見えるものよりも，見えないものや感じるものを描きたい，それが「絵描き」として健全な精神のような気がします。

「巧い」よりも「好い」を目指したい自分としては，やはり進むべき方向は「リアル」ではないのです。と，少しぶったところで今回は終わりにしたいと思います。

都築和彦

Tsuduki Kazuhiko

天使遊戯

制作環境
本体：IBM Aptiva Pentium/200MHz
メモリ：256MB
HDD：4GB
タブレット：WACOM ArtPad
OS：Windows95

使用ソフト
LightEditor

今回は試みとして2DCGイラストの中に3Dを取り入れてみようと思います。3D作成部分には，NewTecのLightWave3Dを使用しました。

ご存じの方が多いと思いますが，LightWave 3DにはBoneという機能があります。こちらの話を少ししておきましょう。

人形を作り，動きをつけたい場合，胴体，手足を，それぞれ別オブジェクトにして動かしてしまうのが簡単ですが，関節部分でどうしてもつなぎめが目立ってしまいます。関節のつなぎめをなくすには，胴体や手足が一体化された人形オブジェクトを作り，関節部分に「Bone」を仕込み，Boneを動かすようにします。

すると，つなぎめなく，ぐにゃりと有機的に変形させることができ，ポーズを自由につけられます。

こう書くとBoneは実に便利な機能のようです

図1 基本となる人体モデル。一体成形されています

図3 ポリゴン数が多すぎるとBoneの影響が思わぬところまで広がってしまう

図2 肩やひじなどにBoneを設定するとこのようにポーズを変えることができる

が，実際は制御がことのほか面倒です。Boneは，隣接したポリゴンのポイントをBoneの動きに従って移動させるわけですが，どこのポイントがどれぐらい影響を受けるかが，制御しにくいわけです。指定されたBoneの影響を受けるかどうか，ポイント単位で指定できるとよいのかもしれませんけど，それはそれでエディットが大変になるでしょうね。

その代わりに，影響を受けさせたくないポイントには，別のBoneを仕込むという対策をとることになります。

想像上の動物なら，動きをいいかげんにつけてもかまわないでしょうが，人間とかになると動きの不自然さはごまかせません。必然的にBoneの数が増えて，それらを管理するのがまた大変なことになります。

図1が今回使用した人形のボディです。全体で5000ポリゴンくらいです。全体がひとつのオブジェクトというわけではありません。羽根は別オブジェクトで，頭も別オブジェクトです。

首のところに，胴体のBoneの支点と，頭のピボットポイントがあります。眼球も別オブジェクトで，動かすことができるようになってます。すべてをBoneで制御するのも大変なので，適当に楽しましょう。

足と手は，Boneを仕込むときに楽なように，垂直，水平にしています。このオブジェクトの場合は，ポリゴンが単純なので，人形の姿のままBoneを入れてみることにしました。Boneの影響範囲を制御しやすいように，あらかじめ手足を引き伸ばしたようにモデリングする手法もあります。

リアルな表現を目指すと制作に時間がかかりすぎるので，「3DCGキャラクター的なデフォルメ」を目指しました。ポリゴンのエッジも3Dならではの味としてとらえ，細分化させていませんし，テクスチャも顔と羽根だけで，服はポリゴン別にサーフェスを変えているだけです。

図2がBoneを関節に使ったようすです。Boneの特性として，関節部分にポリゴンを増やすと，押しつぶされたように変形してしまいます(**図3**)。これは，関節の内側のポリゴンを減らせば回避することができます。

Limited Rangeを有効にし，Minimum RangeとMaximum Rangeをほぼ等しくし，影響範囲は腕の太さくらいにして，影響するポイントを限定させます。ポイントを減らして手作業の力技で制御しようというものです。単純な体型の人形には有効でしょう。

最初，Boneを使うときは関節のポリゴンを増やして滑らかに変形させるのが正解だと思い込んでいたのですが，そうでもないなという感じです。

さて，影響範囲を限定するように設定するわけですが，手順としては，Minimum Rangeと，Maximum Rangeを同じ値にして，シーンで動きをつけたとき，ポリゴンがどう変化するか目で確認します。

変化させたくない部分まで変化するようなとき，Boneの先なら，Rest Lengthを調整してみます。Boneの周囲なら，Minimum Range，Maximum Rangeを小さくしてみます。変化が足りなくなったら，Maximum Rangeを大きくしてみます(Boneには，筋肉をつけたり，不自然な変形を回避するパラメータ設定がありますが，かえって制御が面倒になるので使っていません。あれは，本当に架空の生き物を動かすためのものでしょう)。

ひじは，丸みを維持し，つぶれないようにするため，1個のBoneを入れてあります。肩は，腕を振り回すとかの動きが大きいので，2個のBoneを入れてあります。肩から手首の根元まで，計6個のBoneが仕込んであります。

体全体で，30個ほど使ってます。指を動かすとか大人の人形で肉付きが複雑になると，もっとBoneが増えることになるでしょう。

Boneを入れたあとで，いろいろと各関節を動かしてみます(**図4**)。ポーズによっては，どうしても不自然なしわができたり，体のラインが変になります。

再度Boneを仕込み直すよりは，オブジェクトのポイントの位置，ポリゴンの形をモデラーでエディットして，近くのBoneの影響を受けないように(または受けるように)調整したほうがよいことがあります。

それでも最適な位置をみつけるのは大変なので，妥協できるところを探します。この位置決めは，モデリングよりもはるかに時間がかかりました。まだポリゴンがガチャガチャしてるので，整理すれば体の線ももっと自然になるでしょう。

ちなみに，体の中心には背骨の内側に沿って背中，腹，腰に3つBoneを入れています。体全体で大きく動きをつけるときに使います。この場合，Minimum Rangeを小さく，Maximum Rangeを大きくします。これが本来の使い方でしょう。

Boneの位置が確定したら，影響を受けない部分のポリゴンを増やして滑らかにするといいでしょう。

そうそう，人形を作るときは，指先の「爪」を忘れずに作るようにしましょう。普通の人物画を描く場合でも基本です。

このオブジェクトは，サーフェスを変えているだけですが，あるとないとでは，存在感が違ってきます。

2Dと3Dをなじませるために注意したことについて書いておきます。

まず，光源ですが，イラスト全体は，夕景をイメージしてますので，そんな感じのライティングにします。太陽が奥にあるので，奥から差す光を

300%として強く当てます。反対側に逆光としてオレンジ色の光を50%ほど少し当てます。環境光もオレンジ色です。

羽根の動きを表すのと，2Dに自然に溶け込ませるために，レンダリングで，Motion Blurをかけています。

アルファデータを同時に作るように設定します。2Dペイントツールで，2DCGに貼りつけるときに，アルファデータを使うといいでしょう。

遠くを飛んでる人形については，空気を表現するために，Fogを多くかけています。やはりオレンジ色のFogです。

このような全体がオレンジ色に染まるイラストの場合は，ライティングは比較的楽といえましょう。

ちなみに，レンダリング結果としてポリゴンの変な影ができたり，角が目立った部分は，修正を加えています。なんとなく「ズル」をしているような気分がするのが不思議ですね。いくらでも修正が効くのがCGの強みであるはずなのに。

では，2Dの作業についてです。

私は自作のペイントツール〝きらきら筆〟(LightEDitor)を使っています。LightEDitorで説明しますが，他のツールをお使いのときは，適当に読み替えてください。

図4　さまざまなポーズをつけ，レンダリングを行ったもの。今回はこれを素材として使用します

図5　線画を紙に描いてスキャナで取り込んだものをレイヤーとして，背景に合成させ，背景を描きます

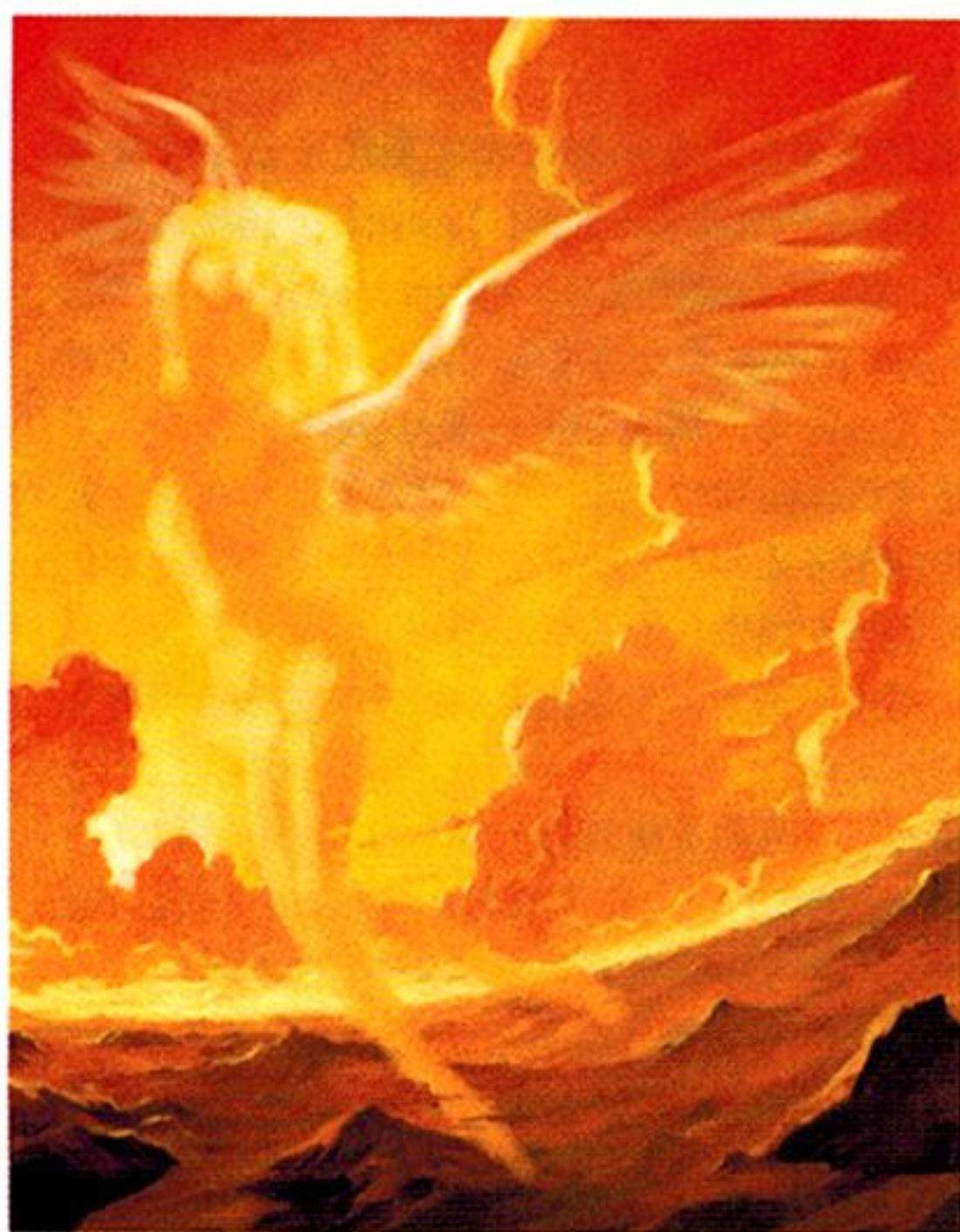



図6　コントラストを上げて色調調整してみました

図7　羽根のデータを作ります。アルファデータも同時に作ります

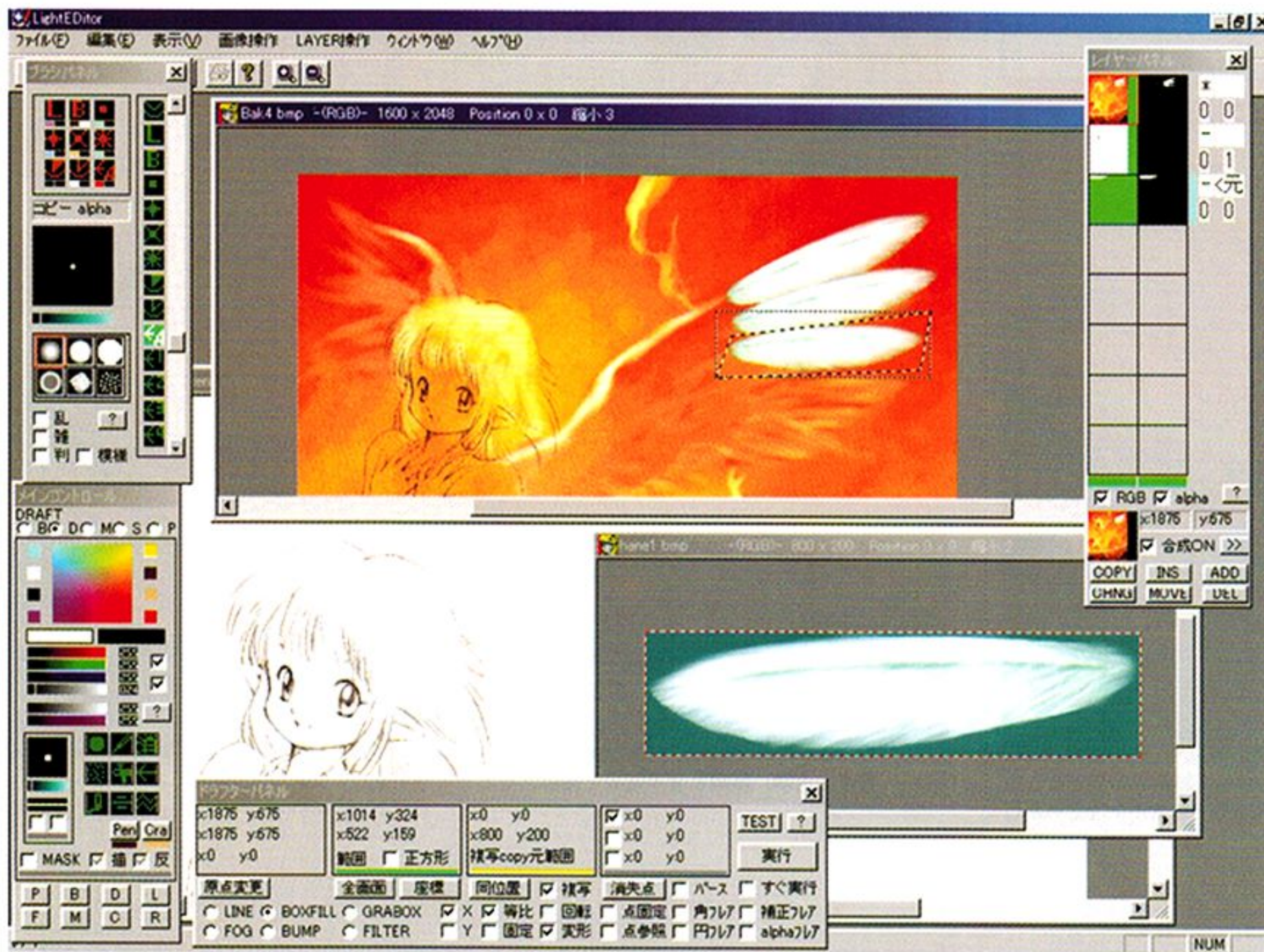



図9　背景に羽根を描き込んだものです

図8　変形コピー機能を使って，背景に貼り付けていきます。羽根のアルファデータを参照すれば，羽根の形に従って，コピーすることができます。LightEDitorでは，「コピー alpha」筆を使います

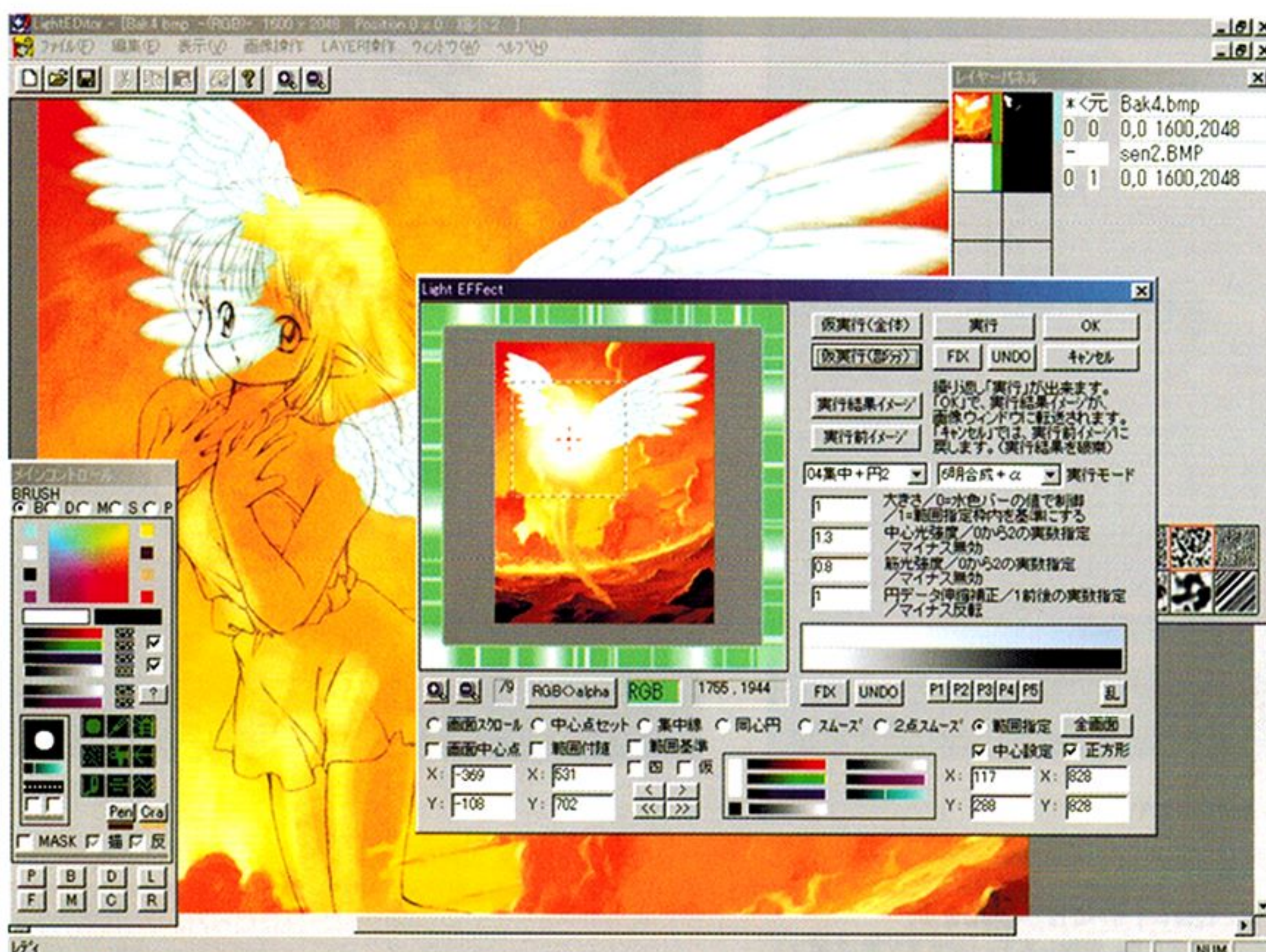



図10　羽根の根元に光輪を描き足します。「LightEFFect」を使っています

図11　人物を描き足しました。線画と背景をレイヤー合成させて作業します。線画の色調にオレンジ色を加えてなじませてみました。「あかるく筆」を使いオレンジ色で線画をなぞります。なぞったところの線画がオレンジ色に明るくなります

図12　背景，人物，線画を合成させて完成させたところです

図13　人物の周りにオーラを描きます。
(1)まず，「コレクションパネル」で，「P5」ボタンを押して，虹のパターンを設定します。
(2)「合成色補正」ボタンをチェックします。
(3)「新規作成」で，黒で塗りつぶされた画像データを作り，レイヤーとして重ねます。
(4)「レイヤーパネル」で，モード変更します。
(5)「光学合成」「合成色補正を使う」をチェックします。そして合成させ，黒い画像レイヤーに白で描画します。
　すると，濃淡が虹のように変化してレイヤー合成されます。オーラの流れを見ながら，描画します。「ひきずり筆」を使ってなじませます。このデータはあとで使います

図14　肩のラインがでっぱりすぎかなあと思って修正しました。奥を飛んでる天使をコピーで貼り付けました

図15　先ほどのオーラを合成しました

図16　さらにオーラを,「合成色補正」ボタンを解除して50%合成させました。すると，全体に明るくなります

図17　手前の天使を描き足しました

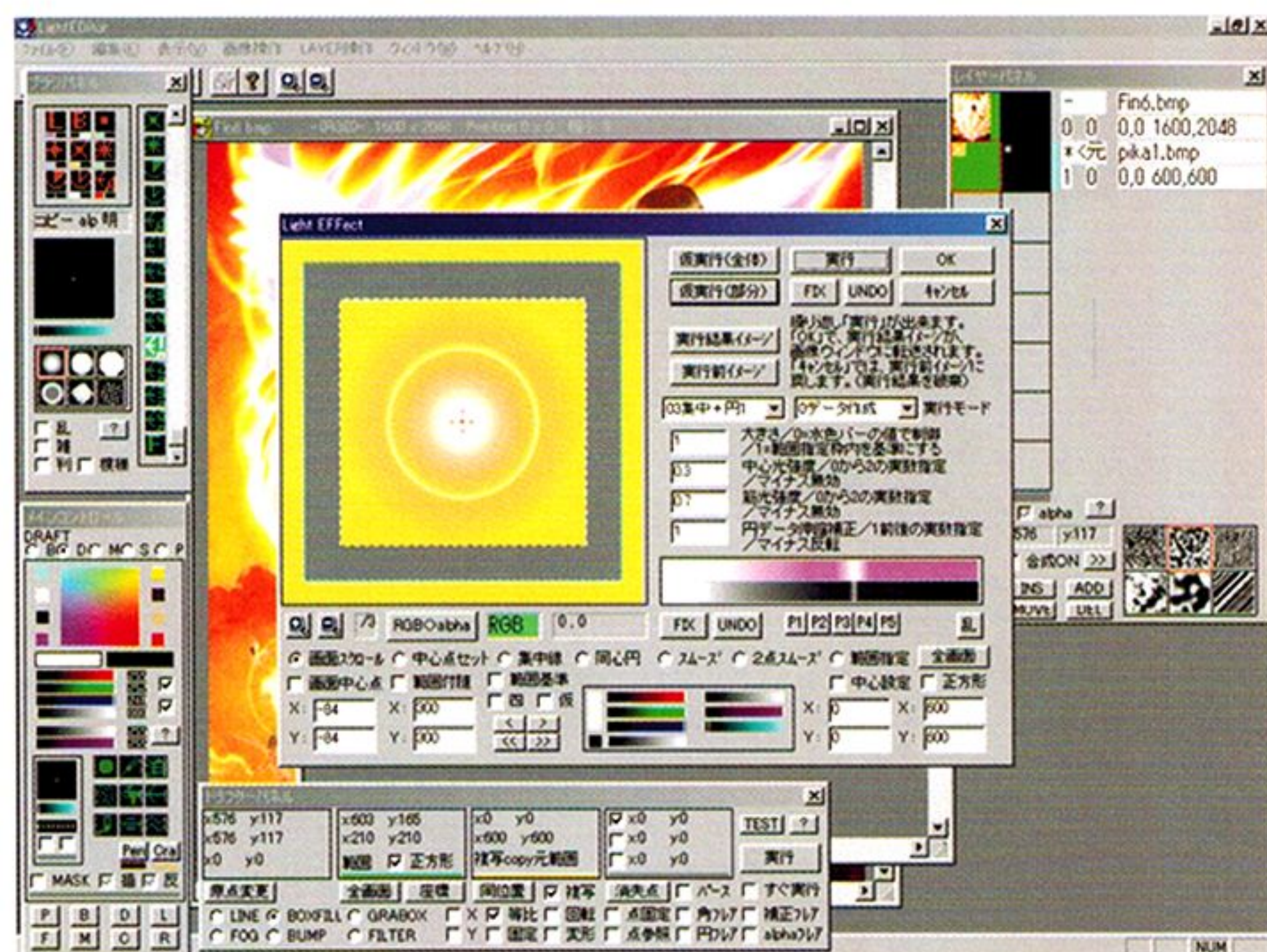
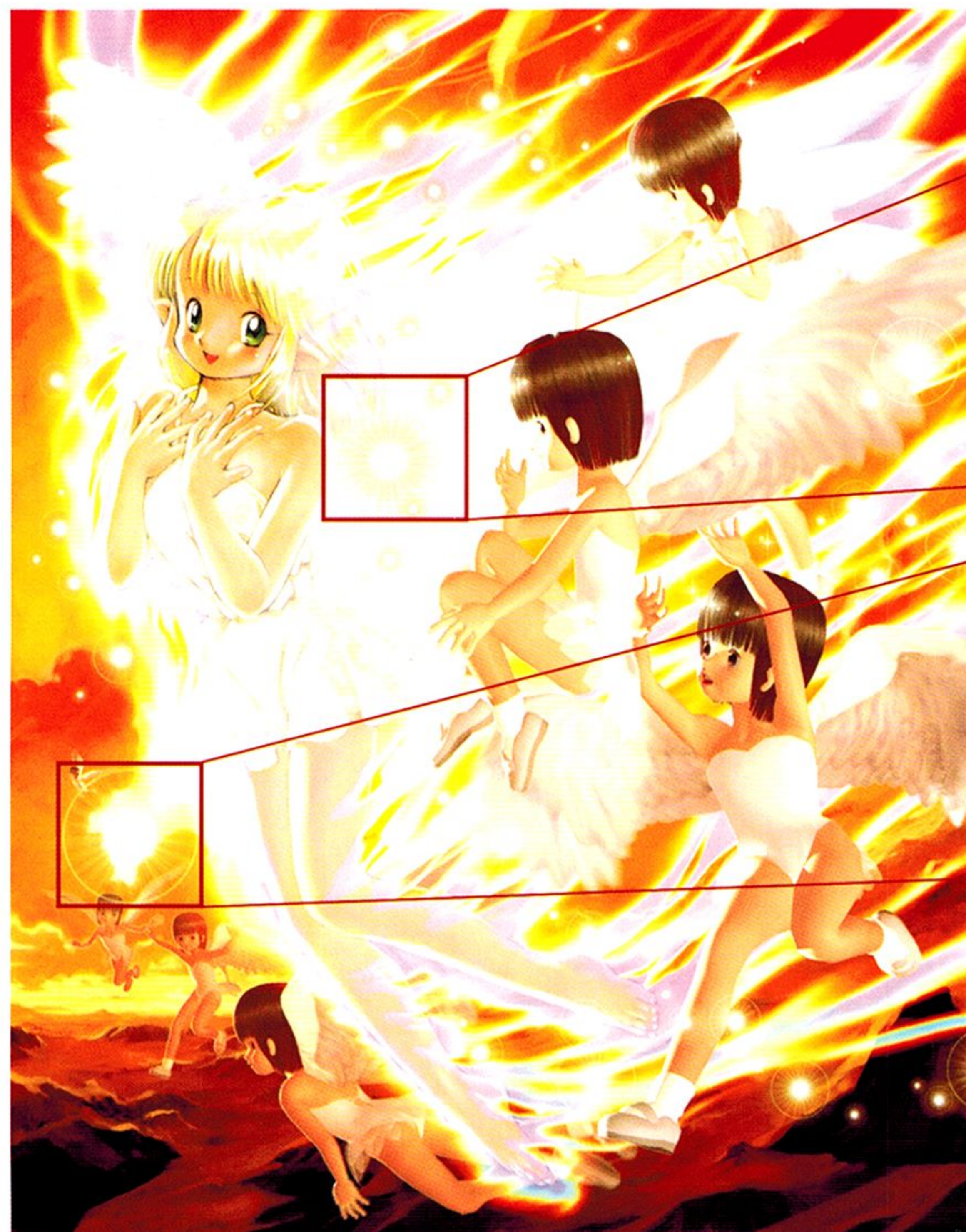
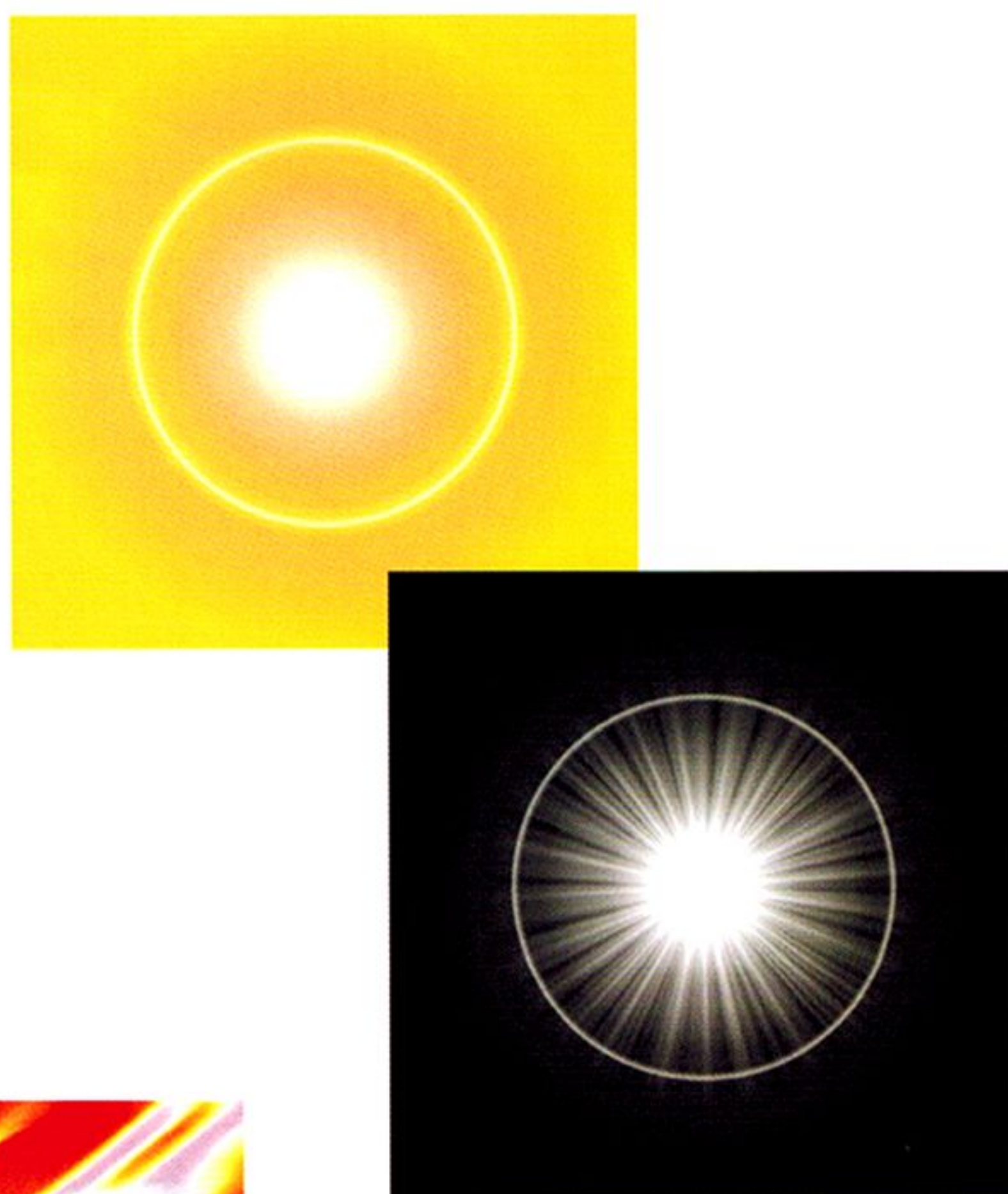



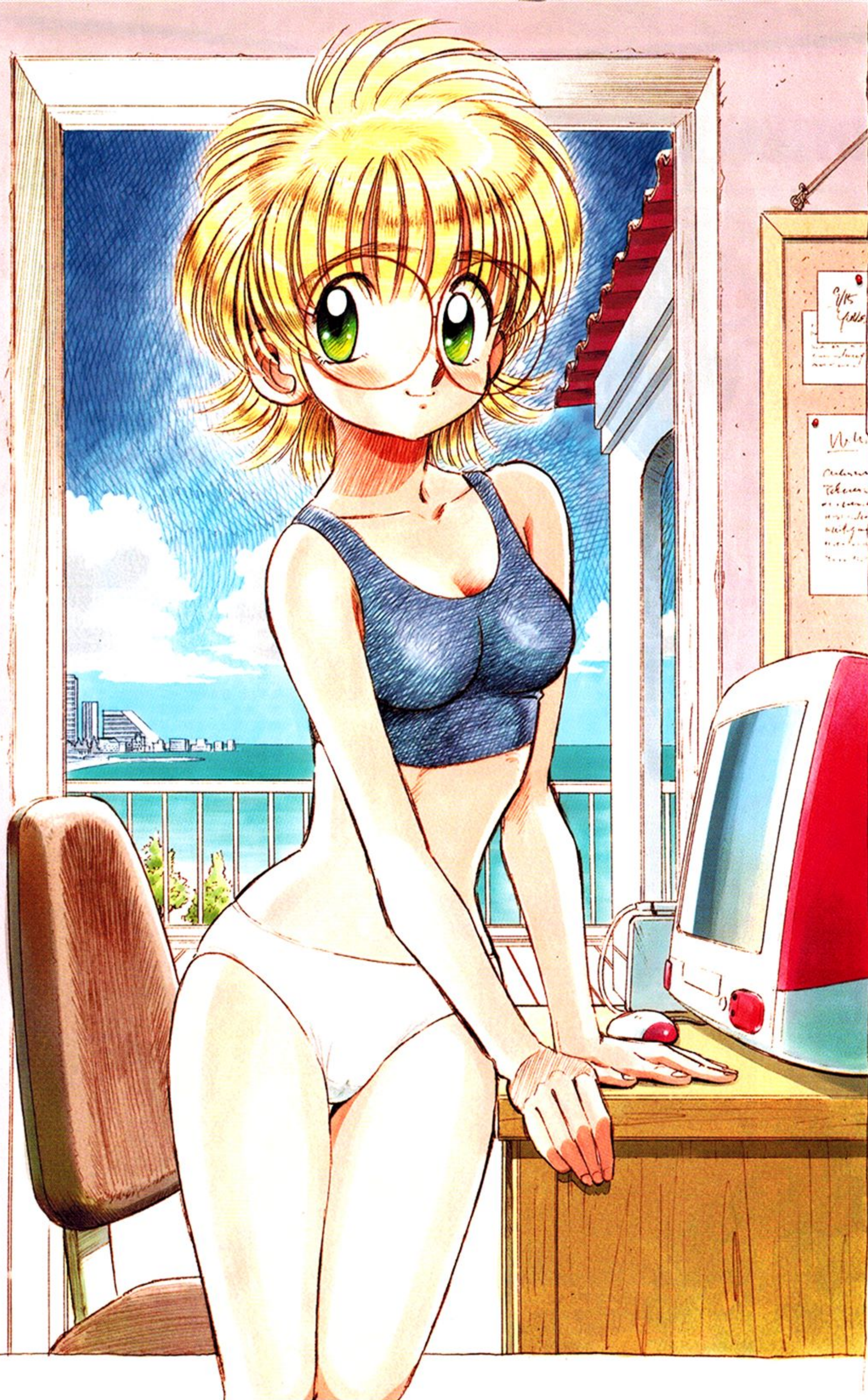
図18 「LightEFFect」を使って，光輪のデータを作ります。アルファデータも同時に作ります

「コピー　alpha」による合成

「コピー　alp 明」による合成

図19 全体に修正を施しつつ，光輪をコピーして貼り付けていきます。「コピー　alpha」と，「コピー　alp 明」合成を使いました。
「コピー　alpha」は，光の色味が残った感じで，アルファデータに従ってコピーされます。「コピー　alp 明」は，背景に光学合成されるようにコピーされます。
　さらに，「きらきら筆」で，部分的にきらきらさせてみました。
　最終的に，全体に赤みをさして色調補正して完成です。
　いかがでしょうか。
　3Dが人物で，2Dの人物を絡ませる場合，存在感の相違をいかに扱うかが難しいですね。3D部分が，物体とか背景とかだともっとやりやすいでしょう。2Dと3Dの組み合わせは，テーマが限定されるでしょうが，チャレンジしがいのあるものだと思いました

末次徹朗

Suetsugu Tetsuro

制作環境
本体：Power Macintosh 8500/120＋208MBメモリ
アクセラレータ：Interware G3 233
タブレット：WACOM UD-0608-A
スキャナ：HP ScanJet IIcx

夢のハワイ

僕の絵はマンガである。だからCGを描く場合も，
紙に鉛筆もしくはペンで線を描き，Painterの水彩筆のみで着彩する。
線画で構成されているマンガ絵には，水彩の質感がいちばん似合うと思うからだ。では始めよう。

下書きをする

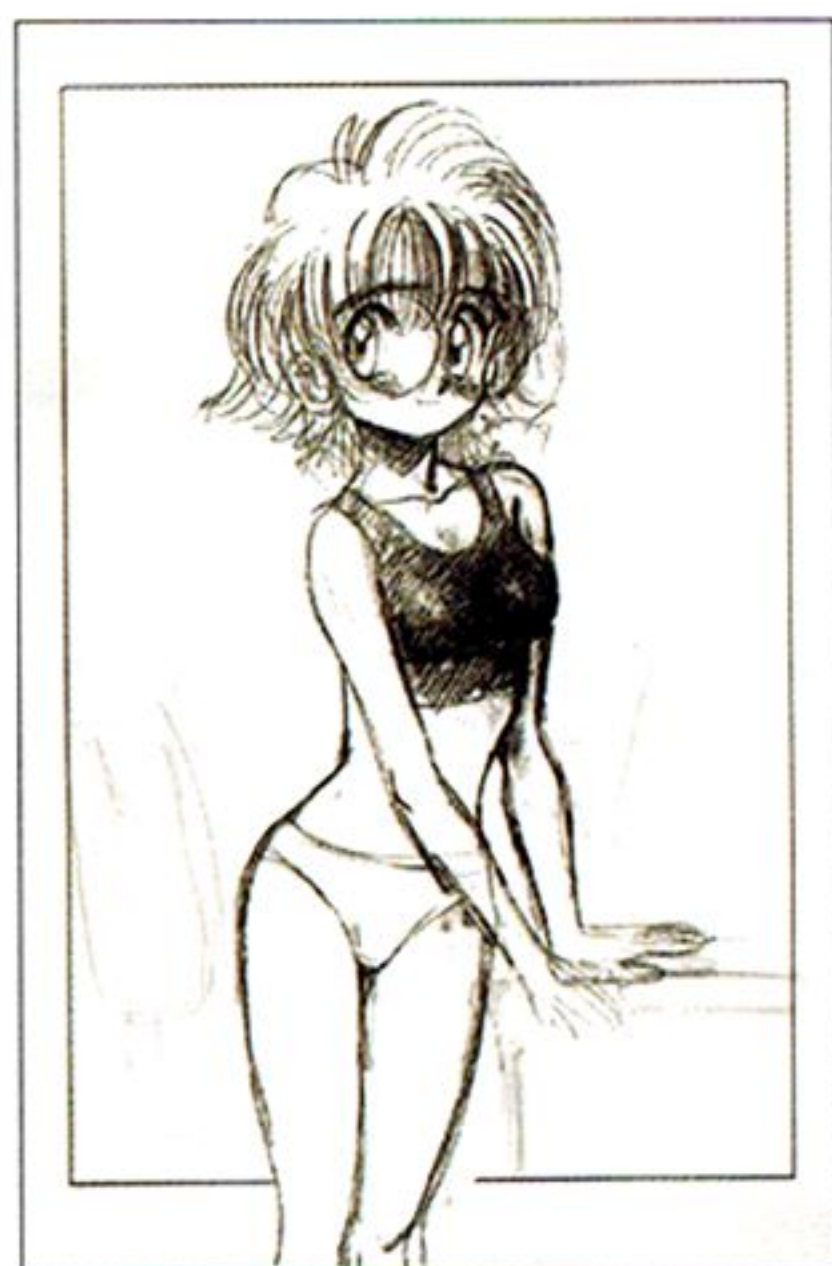



図1　まず紙と鉛筆で下書き。今回はなんにもモチーフが浮かばない。しかし締め切りは迫っている。悩んでいてもしょうがないので，まずは無難に女の子を描く。本来人物を描く場合，どういう人なのか，年齢や性格や仕事や趣味などイメージしていくと，それなりに情景やドラマ性も生まれてくるものだが，しかし，いまはなにも思いつかない。
　苦し紛れに描いていたら，なぜか下着姿になった。うーむ，まあよしとするか。我ながらじつにいいかげんである。人物を描いたあと，背景や室内になにがあるか，アタリを取りながら考えている。ここはどこ？　この娘は誰？　なぜに下着？　さっぱりわからないぞ。ひょっとしたらこの絵は失敗作に終わるのか？

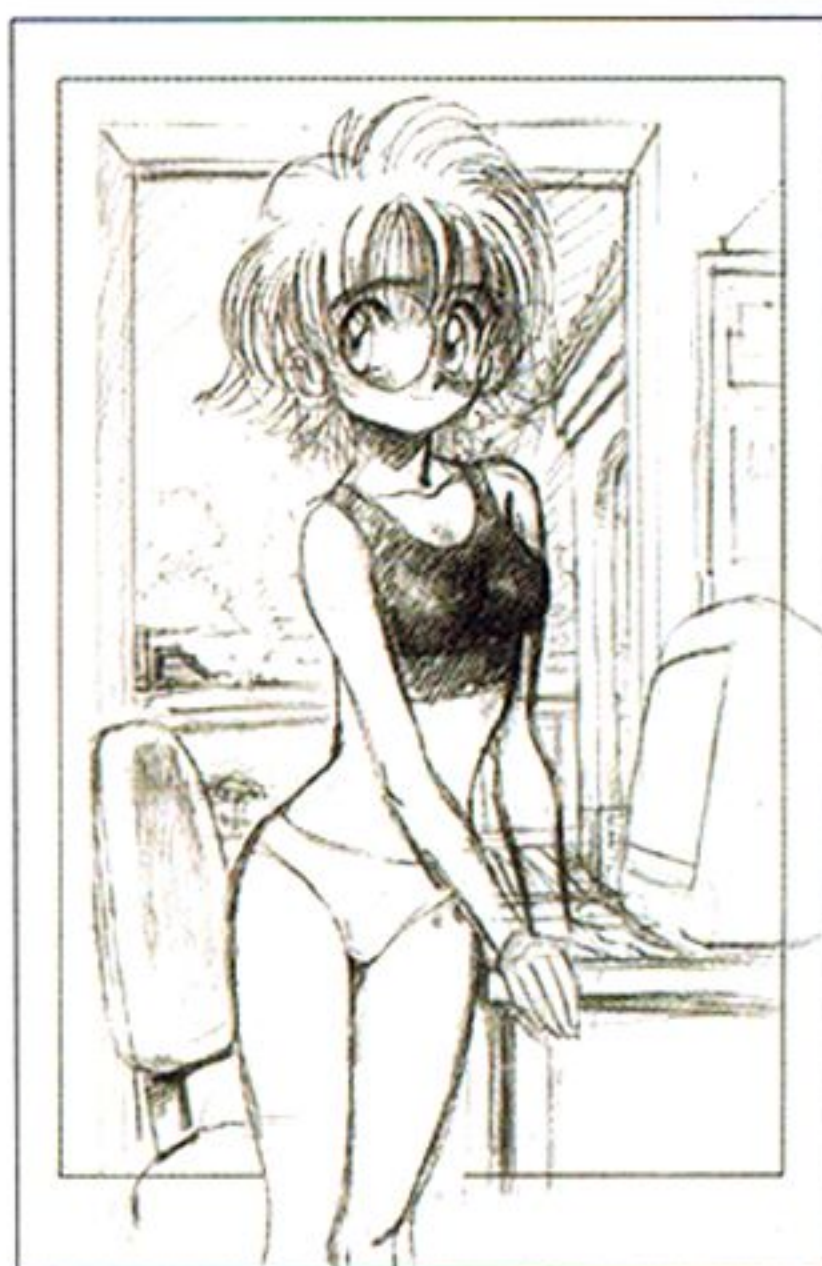



図2　ふと壁にドアを開けたほうがいいように思った。外には空。一気に画面に奥行きが出る。海も見えたほうがいいね。あ，わかったぞ，きっとここはハワイだ。じゃあ，ここは僕のハワイオフィスで，この娘は僕の現地秘書だ。アメリカ娘だ。なるほど，だから下着なのか。ハワイだもんな。……なにやら怪しげな妄想がふくらんで，がぜん気持ちが乗ってきた末次である。マンガなんてそんなものである

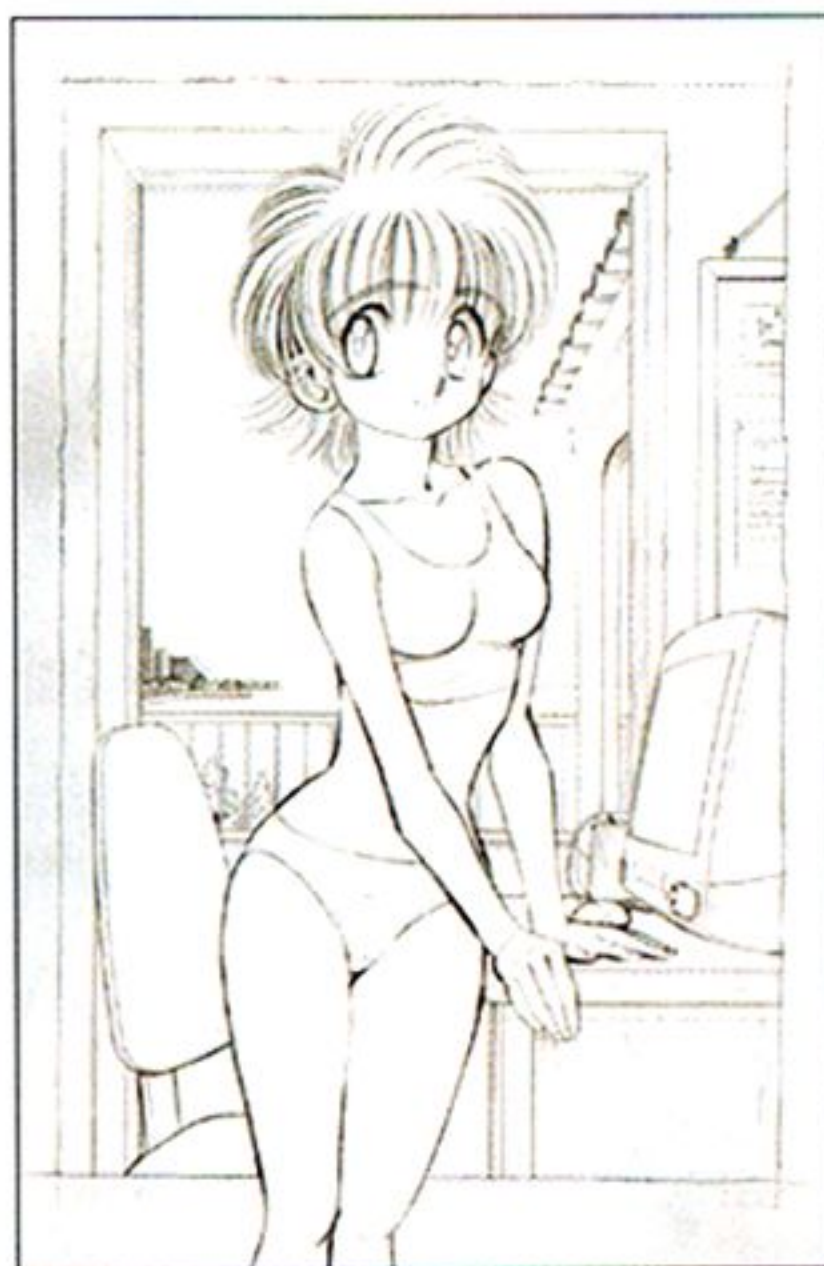



図3　ラフの下書きを，ライトテーブルを使ってトレスする。トレスはシャープペンを使う。必要なところは定規を使って綺麗に

図4　清書した絵をコピーして，その上からさらにシャープペンでタッチを入れていく。こうすると先ほどの線は黒に，タッチの線はグレーになる。こうしておくとスキャンしたときに濃度差がついて，都合がいいのだ。今回の描き方は，とにかく線で画面を構成し，あとで軽く色を乗せる手法。基本的にマンガ原稿の着彩と同じやり方である。タッチの線や背景の線はとても多いので，ちょっと目には大変そうに見えるかもしれないが，結局紙に鉛筆で絵を描く速度は，ほかのどんな画材より速い。線画で下準備を綿密にしておけば，あとの着彩が楽になる。つまりぶっちゃけた話，適当に色を塗ってもそこそこ「画面がもつ」のだ。これはスピード重視のやり方である。マンガの仕事の場合，スピードはなにより重要だ。最低限の処理と時間で最大限の効果を作る。これがマンガの理想である

線画をスキャンする

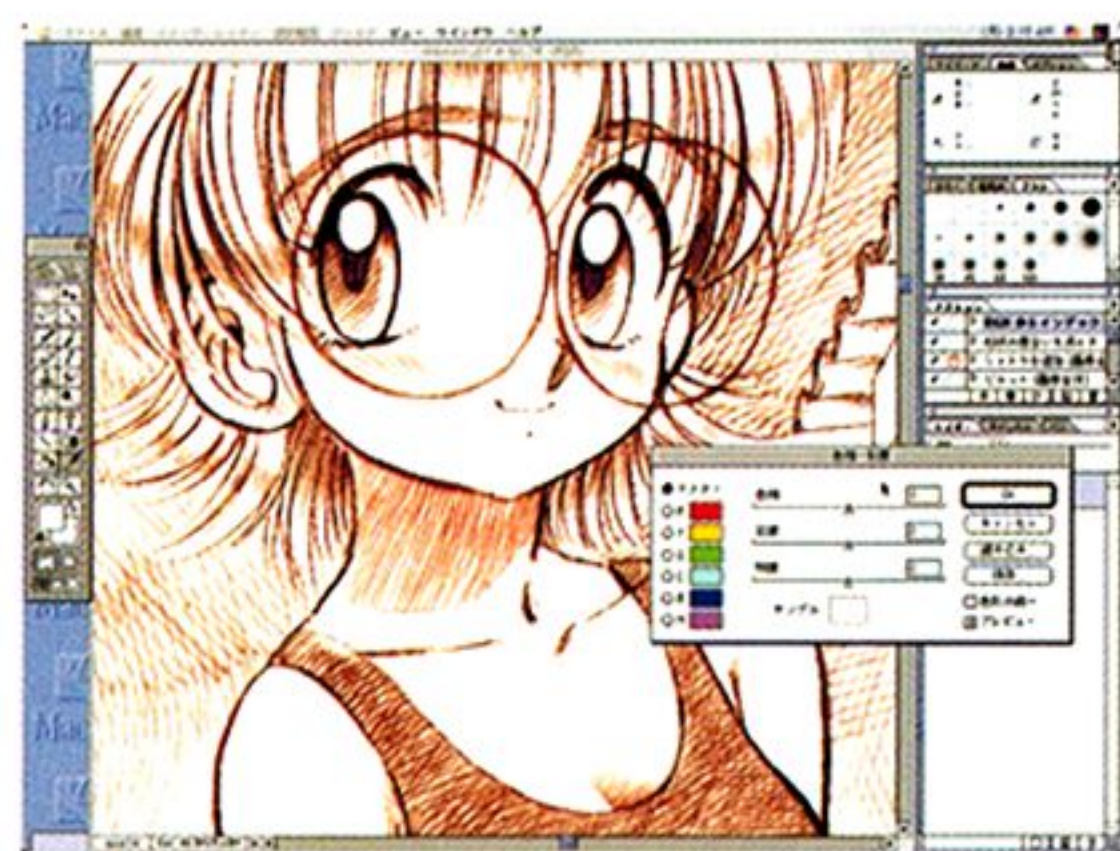



図5　スキャンした線画は，Photoshopを使って，黒から柔らかい茶系に変える。さらに頬のタッチを，明度を上げて薄くしておく。首の下や，胸の谷間のタッチも同様。これは別に，線画を別レイヤーにして着彩のあとで濃度調整してもよいのだが，そうするとデータサイズが大きくなって処理が重くなる。やはりシンプルイズベストなので，あらかじめ処置をしておく

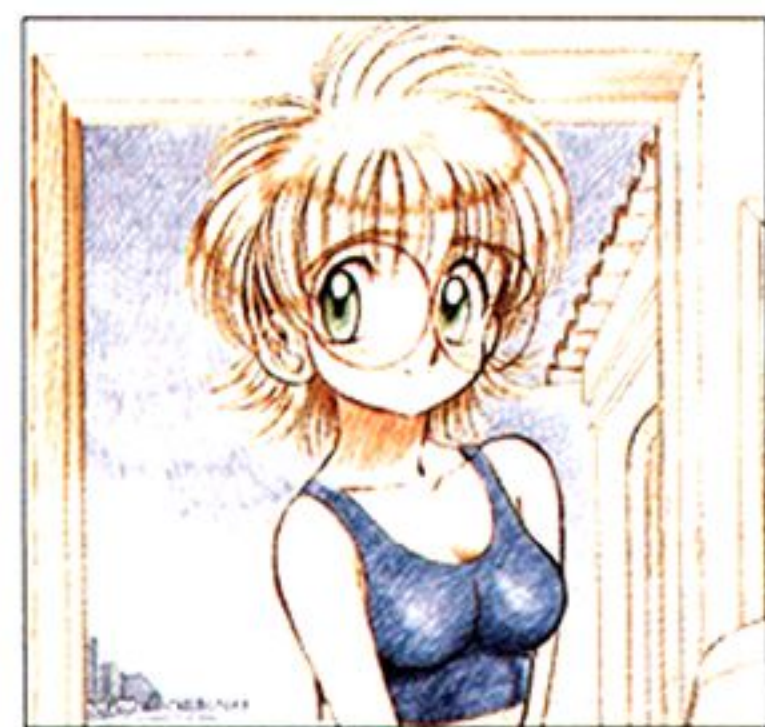



図6　空とブラのタッチを青系に，目を緑系にする。これはもちろん，あとで着彩する色とタッチの色調を統一したいからだ

着彩する

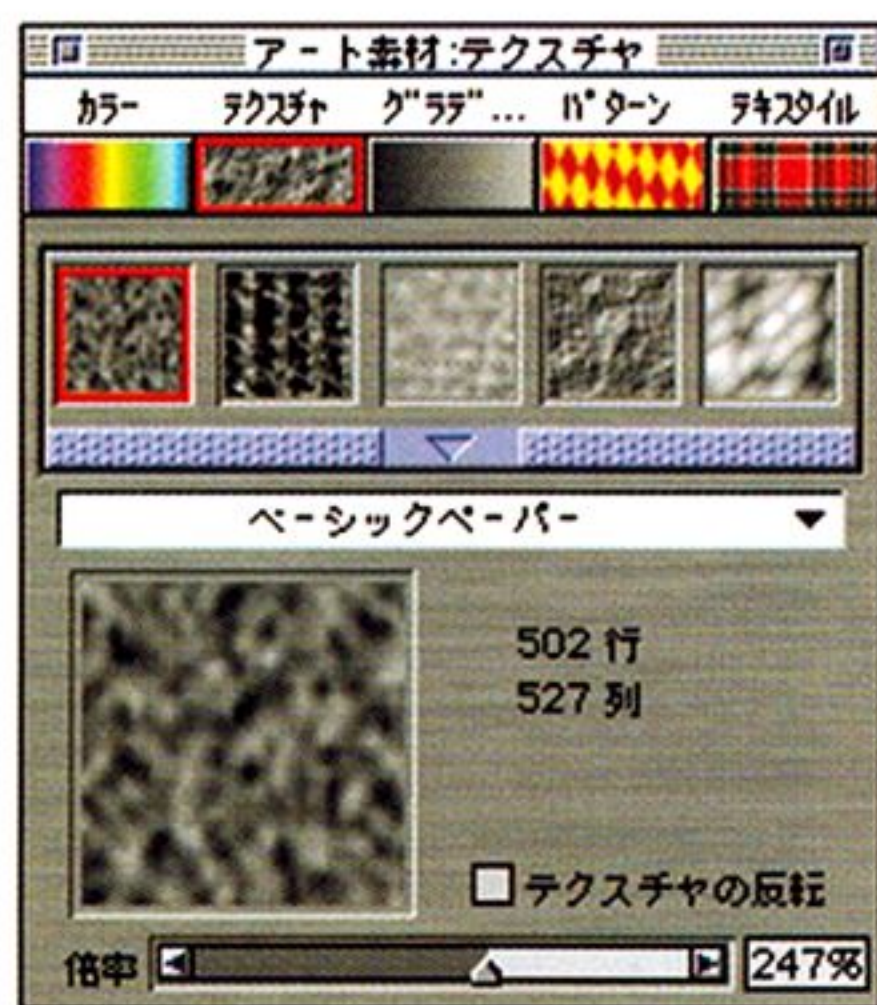



図7　着彩は，Painterの水彩筆のみ。
水彩筆の着彩の場合，非常に大事なのが，ペーパーテクスチャである。これを設定し，画面の塗り色に微妙なムラができることで，水彩の味が出るのだ。今回の絵は，B4の紙に描いた絵を400dpiでスキャンしてあるので，絵のPixel数はかなり大きい。よってテクスチャのスケールも大きくしないと，印刷したときにムラが小さくなり，のっぺりつぶれてしまう。ここではとりあえず250%程度に設定して塗り始めてみた（あとでこれでも少し小さいように思い，300%まで拡大した）

図8　基本色を塗り終わったところ。部分部分を着彩し，はみ出たところは水彩消しゴムで消し，逐一乾燥させながらここまできた。色は空のみ2色を使い，ほかはほぼ単色，ごく一部に影を入れた。線画のテクスチャがあるので，これだけで完成ということにしても，ぎりぎりOKかもしれない

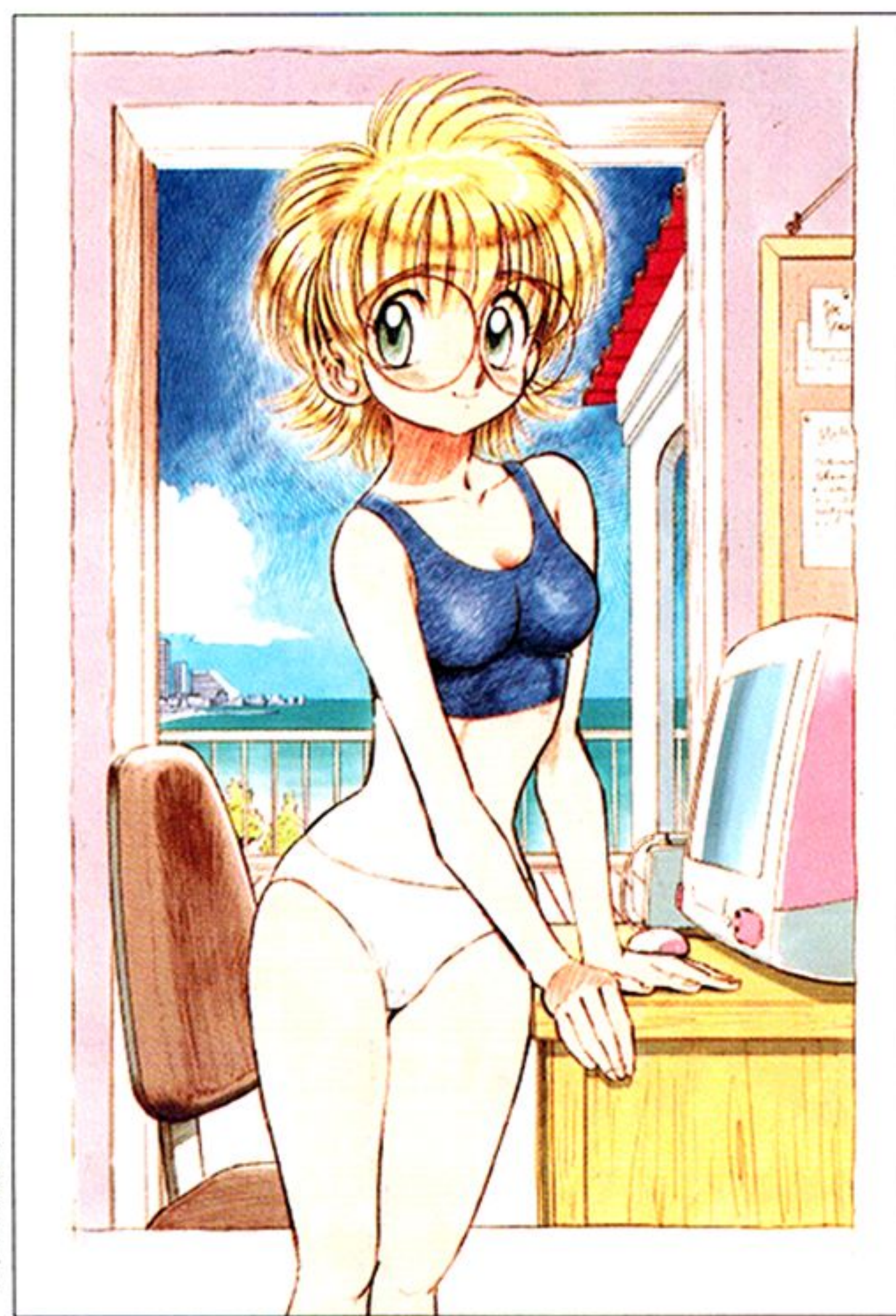
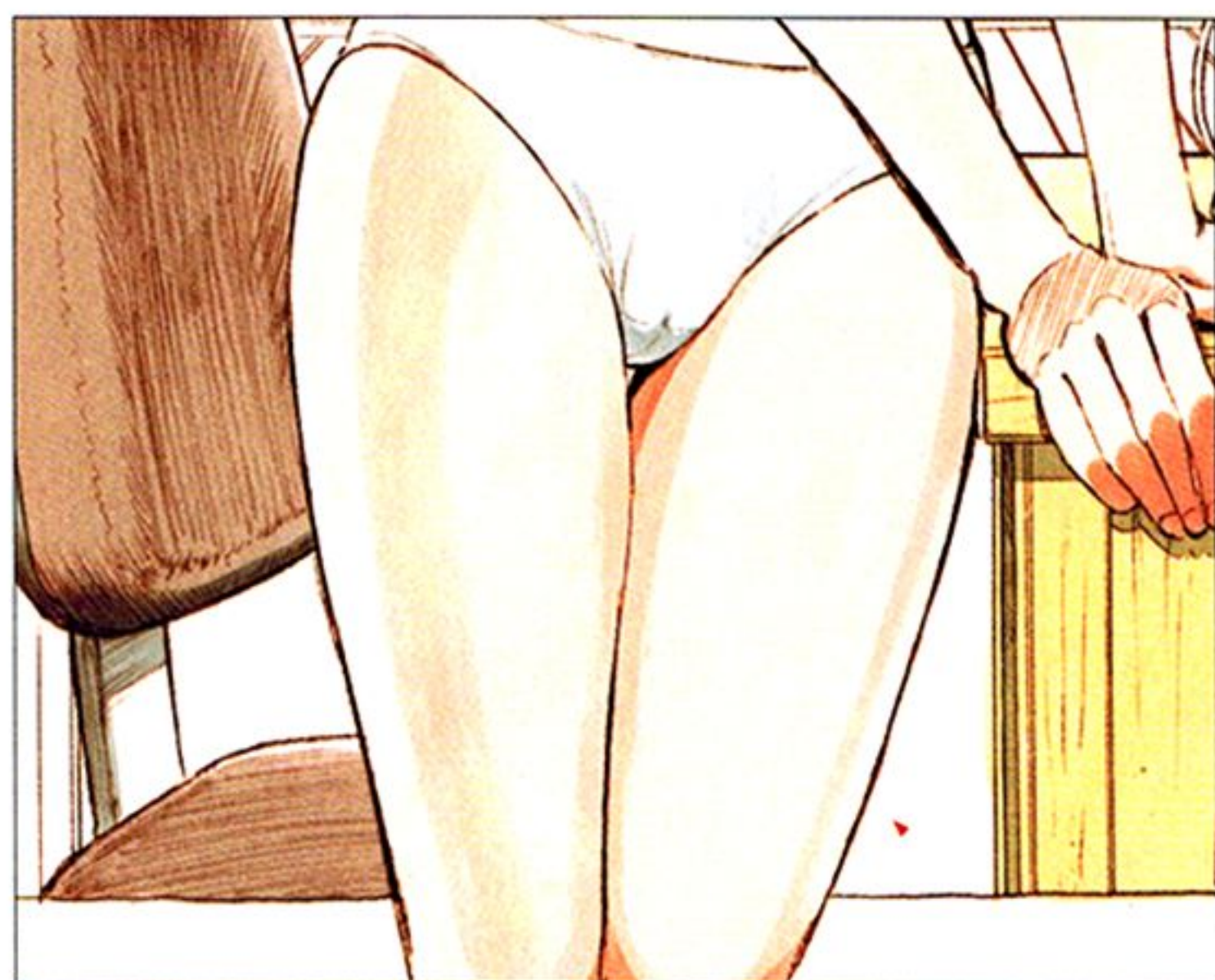



図9　肌に濃い色を重ねる。今回のような水彩着彩の場合，各部分の立体を塗るには，色はそれぞれ最低4色あれば足りると思う。たとえば肌の場合，基本の薄い色，少し濃い色，影の色，わずかに入れる照り返しの色。この4色が基本。ハイライトの部分は白く残すので，正確には5色だ。

もちろんもっと中間の色を乗せていってもよいわけだが，しつこくやるとぬめぬめした感じになって，水彩の透明感や清潔感が消えてしまう。マンガ絵の場合，たとえば身体の立体にしても，筋肉は省略されてイメージとしての立体になっている。だからしつこく立体感を出そうとすると失敗する。極端な話，光源の方向だって気にしなくていいのだ。女の子の絵ならば，女の子がそれっぽく可愛く，おいしそうに描ければそれでよい。らしさとリアルさは違う。こだわってリアルに描いて，マンガとしての柔らかさが消えたり，グロくなったりしては本末転倒である。

なおここの肌にはまだ，照り返しの色は入っていない。肌の色塗りに疲れたら，息抜きにパンツの影を入れたりして楽しもう

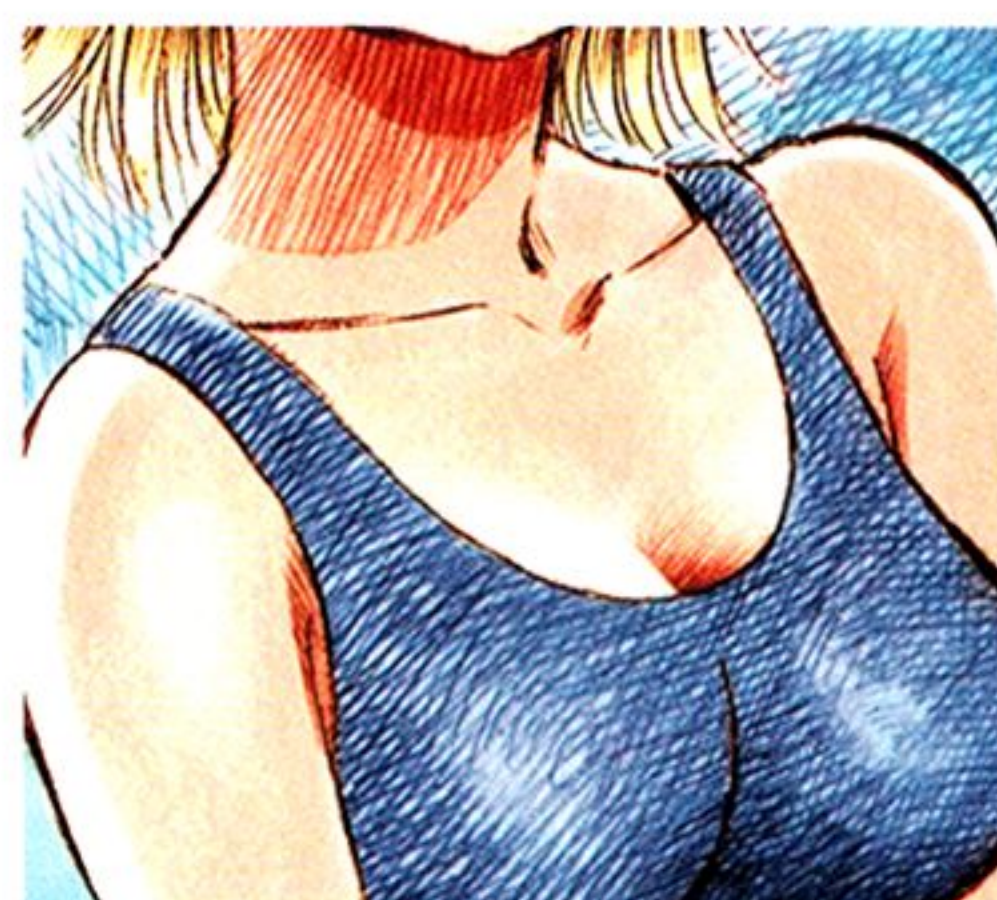



図10　上半身も塗っていく。基本色の上に，少し濃い色を薄くのばして立体を表現する。右肩のふくらみと，鎖骨の下から胸への曲面が，らしく表現できた。2色だけでこれだけできれば上出来だ。首の下と脇の部分には影の色を入れた

図11　基本色に濃い色と影を入れ終わったところ。これが写真なら，外の明るさと室内の明るさが，こんなに等しくなるはずはないのだが，この絵はイメージ画なので気にしないことにする。あとは仕上げを残すのみである

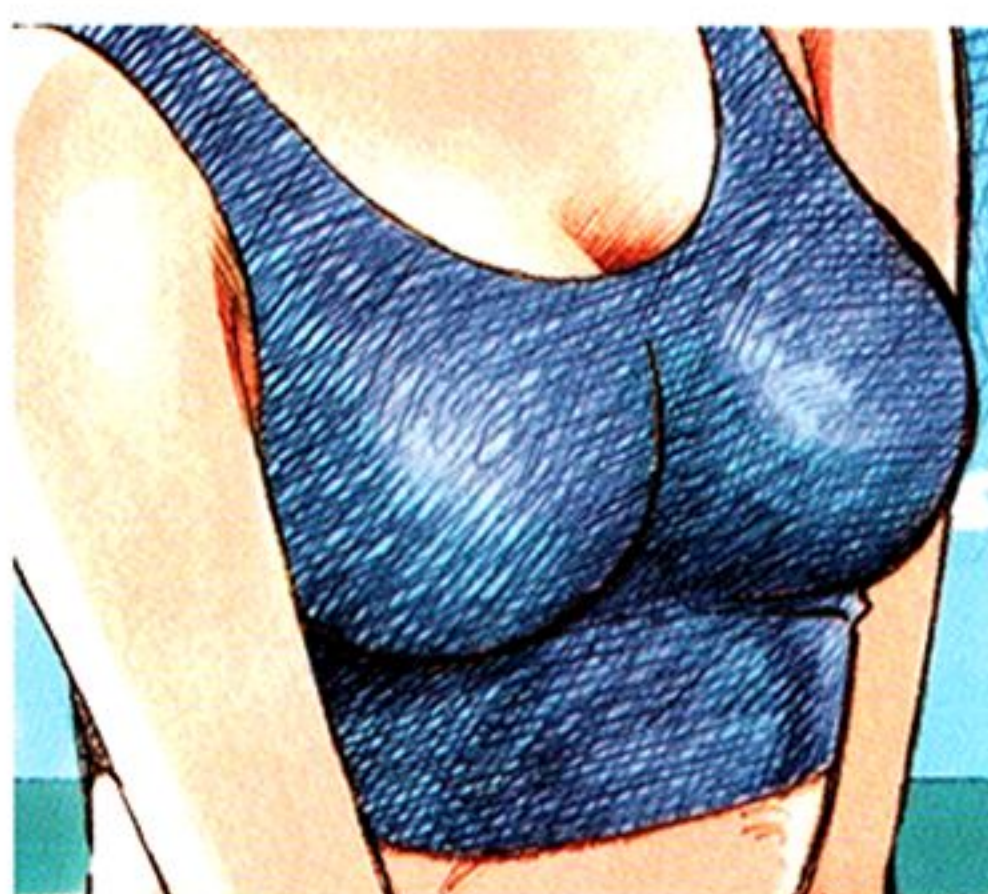



図12　ブラの部分はここまで，あえて基本色一色のままにしておいた。シャープペンで描いたタッチがあるので，必要十分なのだ。ここで初めて照り返しの色を加えてやる。左側に薄い緑を差した。この色は，人物や背景のいたるところに入れていく。照り返しの色は，実際には周囲の色に支配され，いろいろと変わるのだが，薄いブルーやグリーンを入れると絵に清涼感が生まれる。人物の肌に入れる場合は，ごくごく薄めに。入れすぎると汚くなる。照り返しの色を入れ終わると，ほぼ完成だ

図13　最後に目の色を塗る。やはり最後は目を入れて完成とするのが，気分である。目は透き通った立体感が出ればそれでOK。ハイライトはシャープに。ここでも顔の右側や髪の毛に，薄い照り返しの色が入っている。金髪には，髪の色を塗った段階で水彩消しゴムで抜いた，薄い光の帯が入っている

図14　最後の仕上げとして，Photoshopで色味を調整する。右側が調整後の画面だ。iMacの画面を暗く落とし，目の彩度を少し下げ，パンツを白くした。これで完成。時間がある場合は，ここで1日くらい目を休めて再度絵を見直せば，手を入れたいところが見えてくる。「できた」と思ったあと，そこからが最終的な仕上げの段階だという捉え方もある。プロの場合は特にそうだろう。そうすれば確実に絵のクオリティは一段上がる。この絵も，背景やコンピュータの質感など，いくらでも手を入れるところはあるが，それをやると人物の淡い質感が負けてしまうので，あえてこれでよしとした。……しかし，こんなおねーちゃんが僕の秘書になってくれたらなあ。描き終わったあとつくづくそう思ったので，たぶんこの絵は成功である

最後に

僕はPainter1.0の頃から水彩筆ばかりを使っている(ごくたまにチョークも使う)。だからほかの機能はほとんどいらないし，あの分厚いマニュアルを読む必要もない。楽ちんである。

線画と水彩筆のみの表現であっても，絵のクオリティは無限に追求できる。新しいソフトが出るたびにお金を使い，余計な新機能を覚える時間があったら，僕はその分をほかに使う。

また，マンガ絵の魅力，その基本は省略とデフォルメにある。だから，非常に乱暴ないい方をすれば，リアルに描けば描くほどマンガ絵はダメになる。ディテールを描き込めば描き込むほど，つまらない絵になってゆくのだ。

僕は，自分の絵を魅力的にするための努力は少しも惜しまないつもりだが，自分の絵をリアルにするために努力をするぐらいなら，もっとほかのことに神経を使う。もっとシンプルに，もっと可愛く，もっといい構図に，もっとドラマを感じさせるシチュエーションに。

情報過多の時代だからこそ，シンプルで洗練された絵は輝く。願わくば僕は，そういう絵を描きたいと思っている。

森川久志

Hisashi Morikawa

制作環境
本体：PowerMac 7600/132＋BOOSTER 750/233
メモリ：336Mバイト
記憶装置：Caravelle PS-230DX2（MO），Seagate ST-3450/W（HD）
スキャナ：EPSON ES-8000
タブレット：WACOM ArtPadⅡ pro

使用ソフト
Photoshop4.0（途中から5.0）・Painter5・Shade Professional R2（途中からR3）

眺めていて心が安らぐような日常の風景……というのが今回のテーマです。
女の子のカッコがだらしないのは，
裏を返せば目線を送っている相手との気安い関係を表しています。
妹かどうかはご想像にお任せしますが，一応年下の女の子をイメージしています。

図1　まずPhotoShopで下絵を取り込む

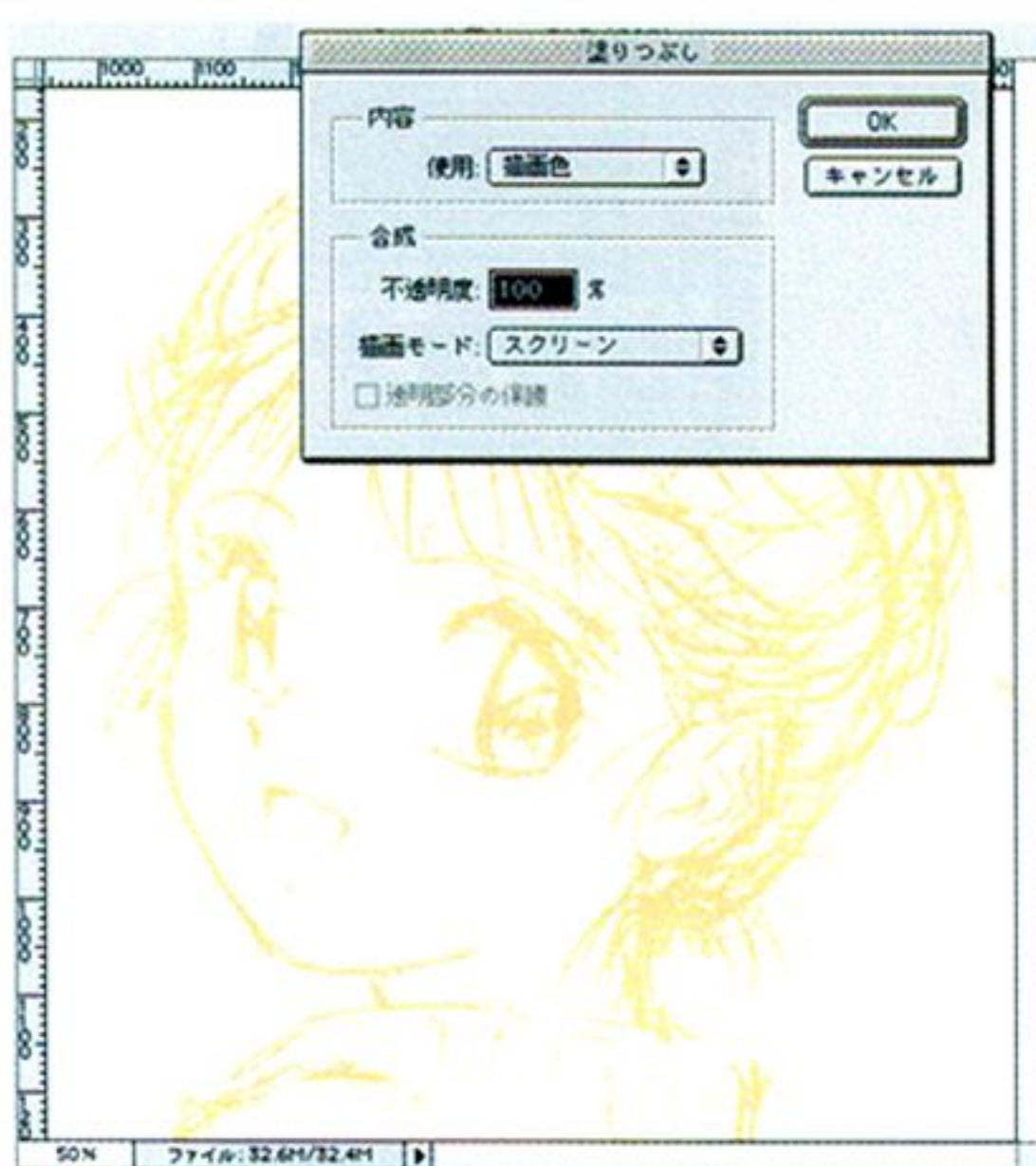



図2　下絵の線を薄い色にいったん変換しておく

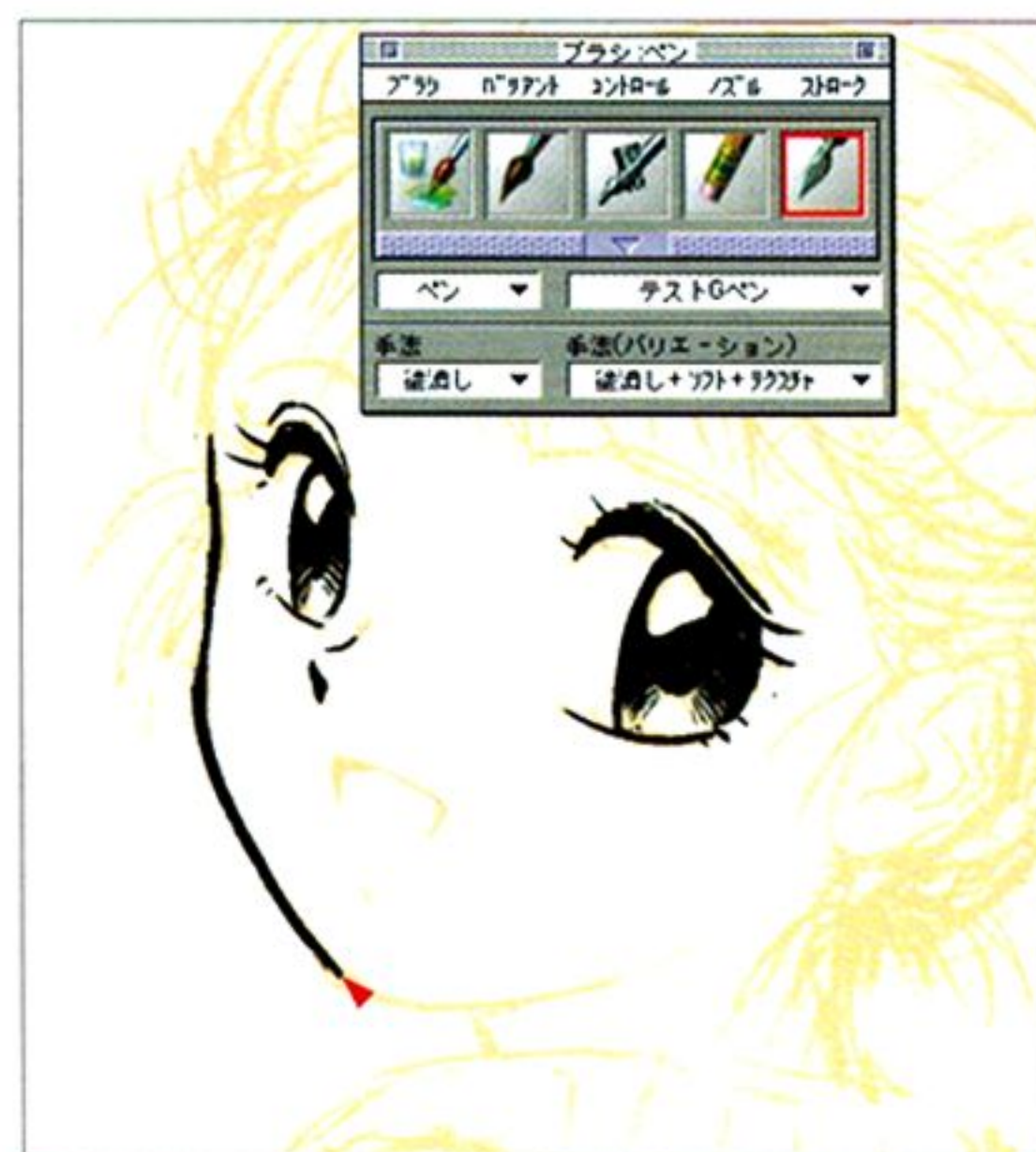



図3　今度はPainterで読み込み，それを下敷きにして主線を入れていく

図4　Painterでのペン表現。上がスムースインクペンで下がスクラッチボードに近い自作のペン設定。実際のペン先に近い表現が可能なことがわかる

図5　具体的なパラメータで比較したところ。左がスムースインクペン。右が設定を変えて作ったペン先だ

人物の制作

鉛筆描きの下絵をスキャナで取り込み，Photoshop上で「塗りつぶし：描画色：スクリーン」で薄い色に変換します(図1，図2)。これをPainterに読み込んで仕上げていくわけですが，Painterでの描画は筆圧感知タブレットを使用することを前提に解説します。

人物の下絵の上から主線を入れていきます(図3)。マンガで使うGペンを使い慣れた人なら，Painterの「ペン：スクラッチボード」が使いやすいでしょう。「ペン」カテゴリーの筆ならストロークの種類を「シングル」にして「ブラシコントロール：サイズ」で筆のサイズをほぼ最小にしてプラスマイナスサイズを最大にし，「上級コントロール：表現設定」でサイズスライダを「筆圧」にあわせればだいたいGペンらしくなります。

私の場合は，もとは「スムースインクペン」だったのが，設定をいじっているうちに結局「スクラッチボード」とほぼ同じ設定になってしまいました。図4は上が「スムースインクペン」，下がその設定をいじって作った主線用のペンで描いた線です。下の線のほうが線の入り・抜きがシャープで筆圧によって太さのコントロールも自在です。

図5は設定の比較。左が「スムースインクペン」，右が改造した主線用のペンの設定です。ペン先の形状は純粋に好みでしょう。Painterの環境設定にあるブラシトラッキングやタブレットのドライバなどの設定で筆圧感知の具合を調節しておくことも大事でしょう。space + optionキーのショートカットで紙を回すように絵を回転させることができるので，どんな角度の線でも引けます(図6)。

ペンを入れたら薄い下絵の線を飛ばします。方法はいろいろありますが，私はPhotoshopの「イメージ：色調補正：レベル補正」を使用しています。Painter上でも「効果：色調処理：明度補正」もしくは「効果：色調処理：明度/コントラスト」などで同様のことができます。

次に人物の塗りに入りますが，塗りの前に「マスク：自動マスク：画像の明るさ」を実行して主線のバックアップを取っておいたほうがいいでしょう。

肌の塗りにはPainterの水彩を使用しています。大雑把に肌の部分を選択しておいてから色を塗り，はみ出した部分は「乾燥」させる前に「水彩消しゴム」で周りを消すという方法を取りました。肌の部分は各部同時に塗り進めていかないと部位ごとに微妙に色が変わってしまいます(図7)。その辺の理屈は本物の水彩画と同じです。

ちなみに私の場合，周りを大雑把に選択する理由は，はみ出した水彩の色を水彩消しゴムで消すときにほとんどバックの白に近い薄い色を消し残したりして，Photoshop上での人物切り抜き時の自動選択に影響を及ぼしても気がつかないことがあるので，どこまではみ出したかはっきりさせて消し残しをなくすためです。厳密なマスクを切らないのは，あくまでも手描きがメインであり，コンピュータでの技は手の延長であることをはっきりさせるためです。でないとコンピュータでの処理の手順の厳密性にこだわりすぎて，かえって時間がかかることになりかねません。

人物の塗りで唯一技法らしい技法といえば，手に持った皿についた洗剤の泡の表現でしょう。ま

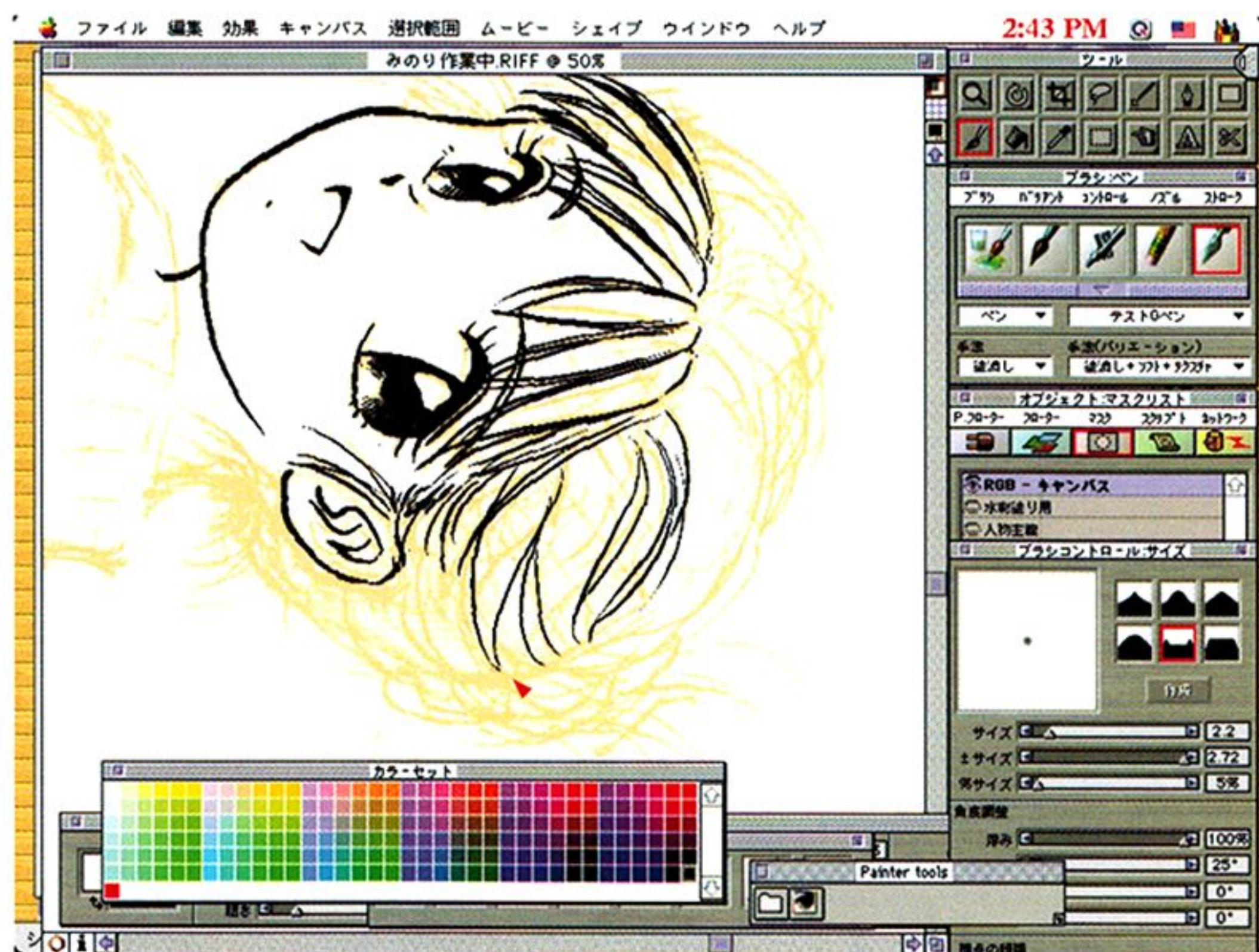



図6　Painterでは紙を自由に回しながら描けるので便利。実際の絵を描くときと同じことはたいていできるようになっているのが嬉しい

図7　複数の部分は均等に塗り進めていかないと，なかなか同じ色になってくれない

図8　スポンジたわしの表現。砂目のテクスチャを加えている

図9　皿に泡をつけていく。小さな泡のなかに大きめの泡を浮き上がらせるため，色を抜く

図10　小さな泡の固まりを吹きつける

図11　大きな泡の部分に縁取りを入れ，テカリを加える

図12　スポンジや手にも同様に泡をつけていく

ずスポンジの裏の部分は基本色を塗ったあとにテクスチャを「砂上テクスチャ」に切り替え，Painter5のプラグインブラシ「フォト：テクスチャ追加」で上からなぞります（図8）。泡の表現はテクスチャを「ローシルクテクスチャ」に変更して，スポンジや皿など泡をつけたい部分を同様に上からなぞります。遠目に見てどう見えるかでテクスチャの拡大率を調整します。

実際の泡を観察すると目に見えない小さな泡の間にところどころ大きな泡があるのがわかります。その大きな泡の部分はだいたい透明で，下が透けたようになっているので，テクスチャブラシでなぞった部分にペンで穴を開けるように地の色（より少し濃い色）で塗りつぶし（図9），それを避けるようにテクスチャブラシの上から白で「エアブラシ：荒目」をかけて泡の固まりを表現します（図10）。そして穴を開けた部分をペンで白く縁取り，泡のてかりを入れればそれっぽくなります（図11）。

手やスポンジも同様にして泡をつけます（図12）。この表現のいいところは単にリアルなだけじゃなくてPainterのブラシ効果やテクスチャなどもちゃんと生かして絵としての表現になっていることだと思います。ただしこの絵では泡が細かすぎて潰れてしまっています。もう少しデフォルメして大きめに描いたほうがいいでしょう。

服の部分も肌と同様の塗り方ですが（図13），衣服の柔らかい材質感を出したかったので，水彩を「乾燥」させて定着させたあと，「ブラシ：水滴」カテゴリーの「水滴（霜大）」で主線を上からなぞります（図14）。だいぶ水彩の柔らかい感じが出てきました（図15）。

最終的には人物にもう少し温かみを持たせたかったので，服の色を暖色系に変更することにしました。Photoshop上で服の部分の選択範囲を作成し，「色調補正：色相・彩度」で色を暖色系に変え，あとはトーンカーブでコントラストを抑え，温かみのある淡い色調にすれば人物は完成です（図16）。……間違えました，こっちです（図17）。

3Dソフトを使用した背景の制作

1　モデリング

背景には3DソフトのShade Professional R2を使用しています。前回は作業をできるだけ単純化するという目的で市販のモデリングデータを使用したため，Shadeはレンダリング機能の使用のみにとどまりましたが，Shadeの本当の醍醐味はその優れたモデリング機能にあります。Illustratorのようにベジエ曲線を使用しての自由曲面のモデリングは比較的絵を描く感覚に近いです。

「詰めShade」とはよくいったもんで，動きの制限された駒を操って自分の目的を最短の手順で達成する……そのために知恵を絞るのが詰め将棋の醍醐味ですが，Shadeで作品を作る作業はこれに

図13　とりあえず，肌の塗り方と同様に塗っていく

図15　だいぶ柔らかい感じになった

図16　間違い……だそうです

図14　水滴を使って水彩らしさを加えていく

図17　色調補正で全体の色調を暖色系に整えて人物部分のでき上がり

よく似ています。Shadeの世界では「そうか，こういう方法があったか！」といった目からウロコが落ちるような発想のモデリング技法に出合うことが少なくありません。プラモデルを組み立てる感覚にも近いShadeのモデリングは，やればやるほどハマっていきます。といっても今回のほとんどのオブジェクトは回転体や掃引（そういん＝ほかの3Dソフトでいうところのパススィープ。単純に一方向へ引っ張るのとパスに沿って引っ張るのと2種類の方法がある）を使った単純な形状で作成でき，Shade得意の自由曲面がものをいうのは，せいぜい流しの部分と湯沸かし器の前面くらいのものです。流し台の水槽部分はこの絵の角度からは見えないのでわざわざ作る意味はなかったのですが，Shadeのモデリングの柔軟性を説明するのにちょうどいいので少し解説してみましょう。

流しの金属部をひと続きのパーツで作ることを考えた場合，上から見た四角形を上下方向に引き伸ばしたほうが，ほぼ同じ形の連続になるので都合がいいと判断しました。とはいえ，流し台の外枠から水槽の中まで一気に引き伸ばすとなると，引き伸ばす方向のラインの形が複雑になるので，やり方としては多少雑な気もします。それでもや

ってみたら，たいした苦もなく綺麗な形ができあがってしまいました(図18，図19)。引き伸ばす方向のラインは最終的に一点に収束する形で面を閉じます(排水口はさすがに最初から考慮に入れてません)。このように大雑把なやり方も許容してくれるあたりがShadeの魅力だと思います(図20)。

あとで気がついたのですが，流しの縁部分を断面で描いてから流し台の縁に沿って掃引し，同様に流しの水槽部分も断面を穴の形に沿って掃引するほうがやり方としてはスマートなようです(図21，図22)。

この場合，流し台上面部と流し底面部の平面は別パーツで作ります。掃引してできた「閉じた線形状」を同位置にコピーしてブラウザ上で自由曲面の外に出すだけです(図23＝この場合の矢印はコピーして自由曲面の外に放り出すという意味)。それだけで面が閉じて穴を塞ぎます（図は別パーツであることがわかるようにわざと離しています)。厳密には流し上面部のほうは外枠と穴になる部分の2つの形状を新たな自由曲面に放り込みます。それで閉じた面と穴の開いた面になります(図24)。

なお，このやり方は食器入れなどに応用しています(図25，図26)。

このようにShadeでのモデリングは開いた線形状・閉じた線形状・円・長方形などの二次元の基本形状を掃引，あるいは回転したものを，さらに変形させる必要があれば自由曲面に変換し，その内部の線形状などを変形したり，ブラウザ上で操作して新たな形状のベースにしたりなどして発展させていく……といったやり方が基本となります。

R3からはプラグイン形式によってさらに便利なツールが増えていますが，基本は変わりません。自由曲面内の線形状を「記憶」させ，その線形状に沿って別の形を「掃引」するという方法を効果的に使えるようになれば，それだけでもたいていの形が作れるようになります(図27，図28，図29)。

このほかに複雑そうな形としては湯沸かし器の水供給用の蛇腹パイプがありますが，これはモデリングではなく表面材質設定のバンプマッピングで実現しています。バンプマッピングはグレイスケール画像を貼り付けてその濃淡でオブジェクト表面の凹凸を表現するものです。拡大すれば実際の蛇腹パイプとは似ても似つかぬものであることがわかると思います(図30)。さらに横着して同じバンプマップをゴムホースの縦縞にも使用しています(図31)。

2 レンダリング

モデリング機能以外のShadeの利点としてはレンダリング画像の綺麗さとあおり補正機能の存在が挙げられます。レンダリング画像のシャープさと細かさのおかげでPhotoshop上でコミック調に加工するときにはエッジの線を綺麗に拾ってくれますし，「あおりチェックボックス」をチェックすることで上下方向のパースを殺したレンダリングをしてくれるのは二点透視法が多いコミックの背景にとって，実にありがたい機能です。もちろん，ただチェックすればいいというものではなく，望遠レンズとあわせて使用するなど違和感なく見せる工夫が必要ですが……。

ここからいよいよスタジオ撮影に入ります。これまでにモデリングしたオブジェクトを全部ひとつのファイルに読み込んで台所のセットを組み上げます(図32)。次に室内の照明を配置して光源設定を行います(図33，図34)。レンダリング時間のことを考えると1灯でも少ないほうがいいの

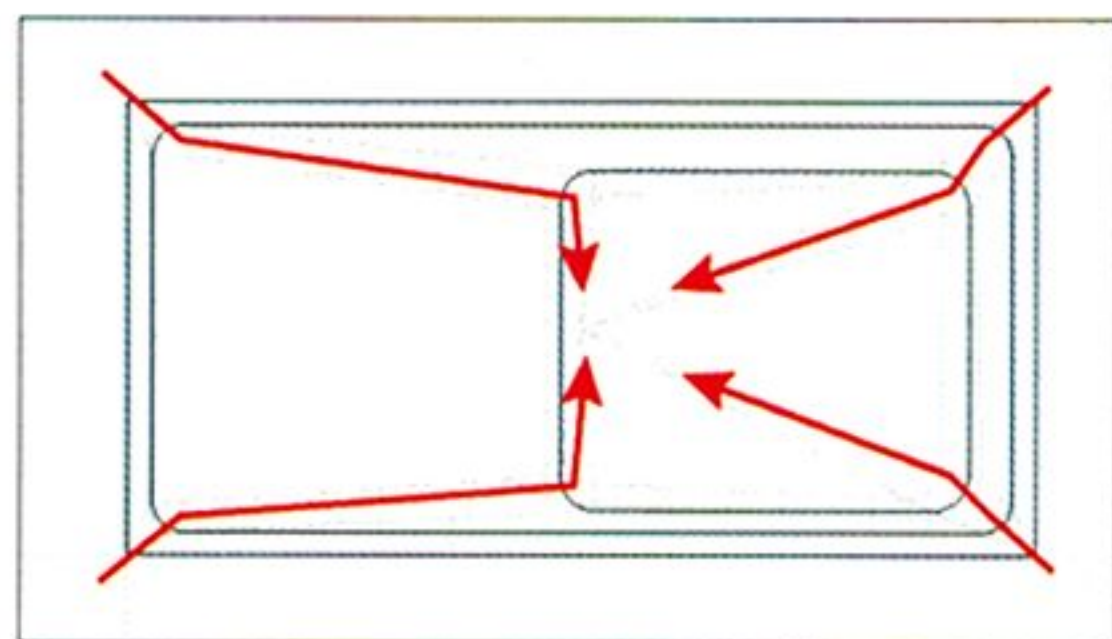



図18　シンク部分を掃引で引っ張ったところ。これだけでもかなり使える形ができあがる

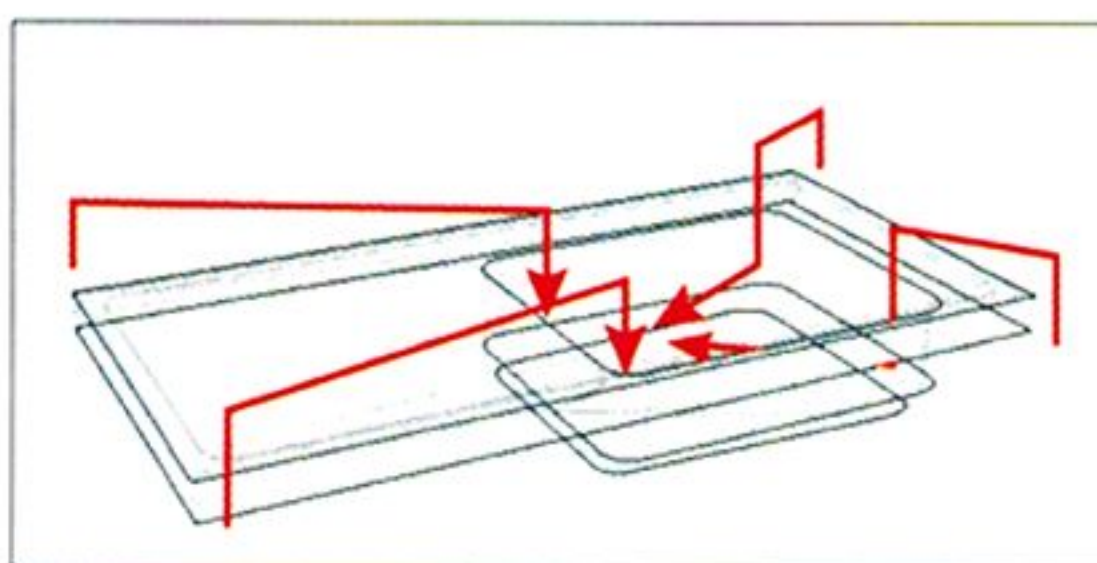



図19　同じく透視図

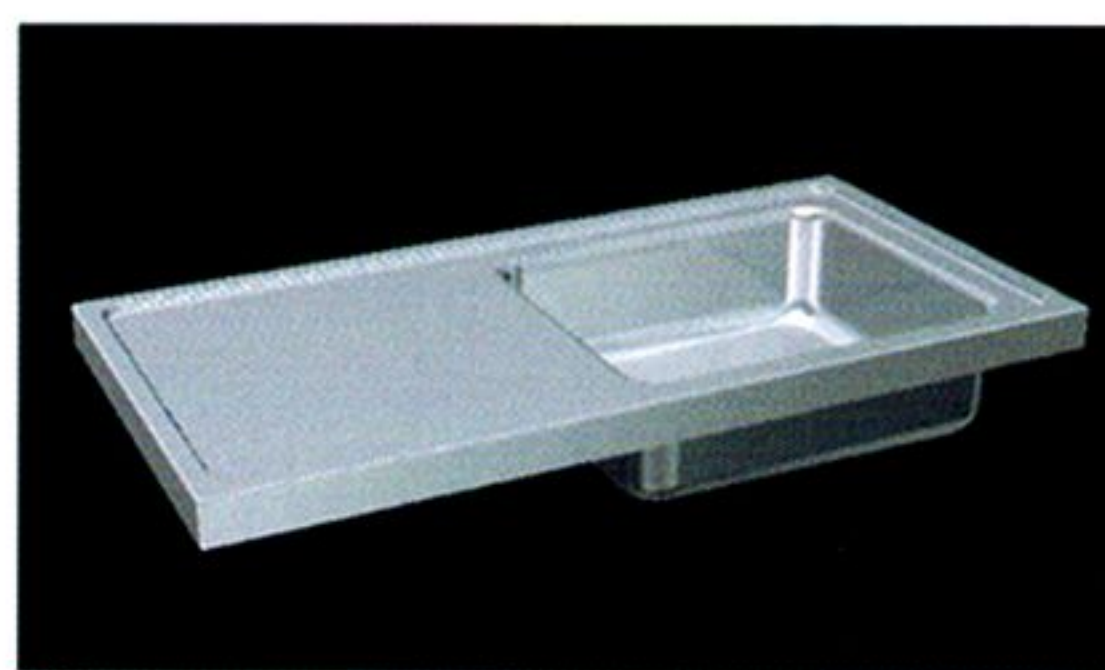



図20　レンダリングするとこんな感じになる。どこから見ても流し台だ

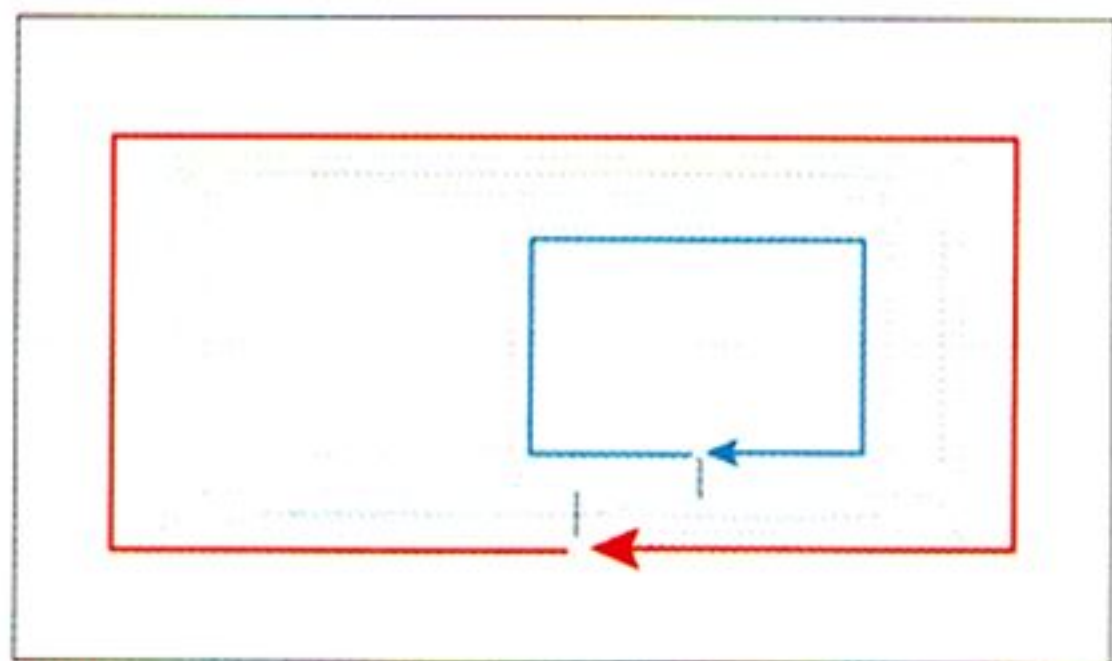



図21　別のパスで掃引した例。こちらのほうがわかりやすそう

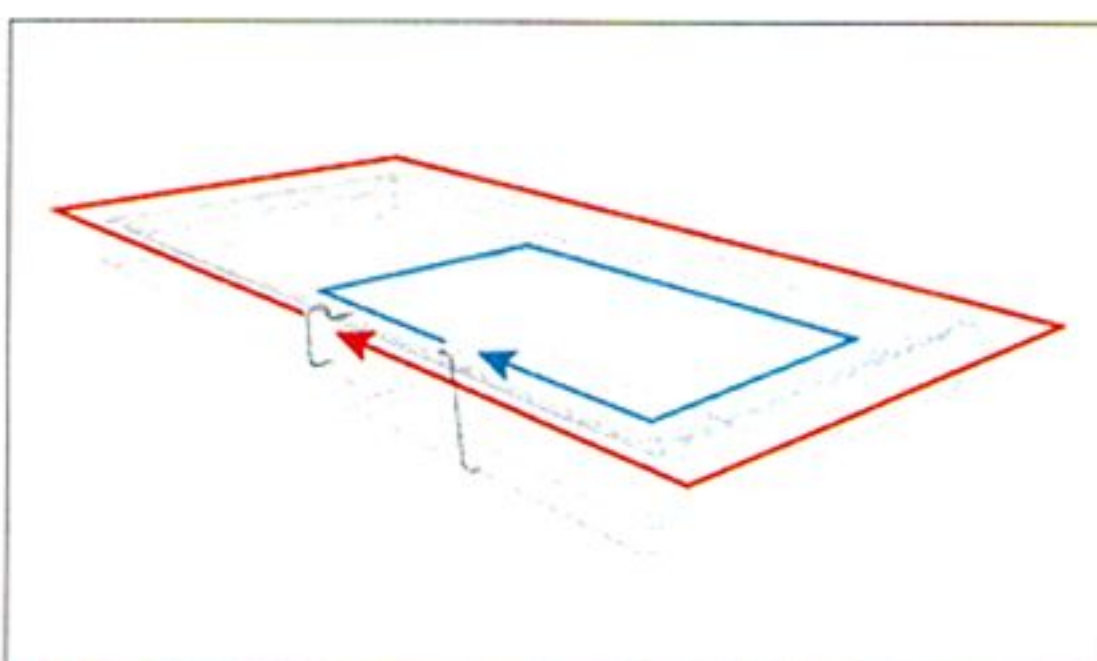



図22　同じく透視図

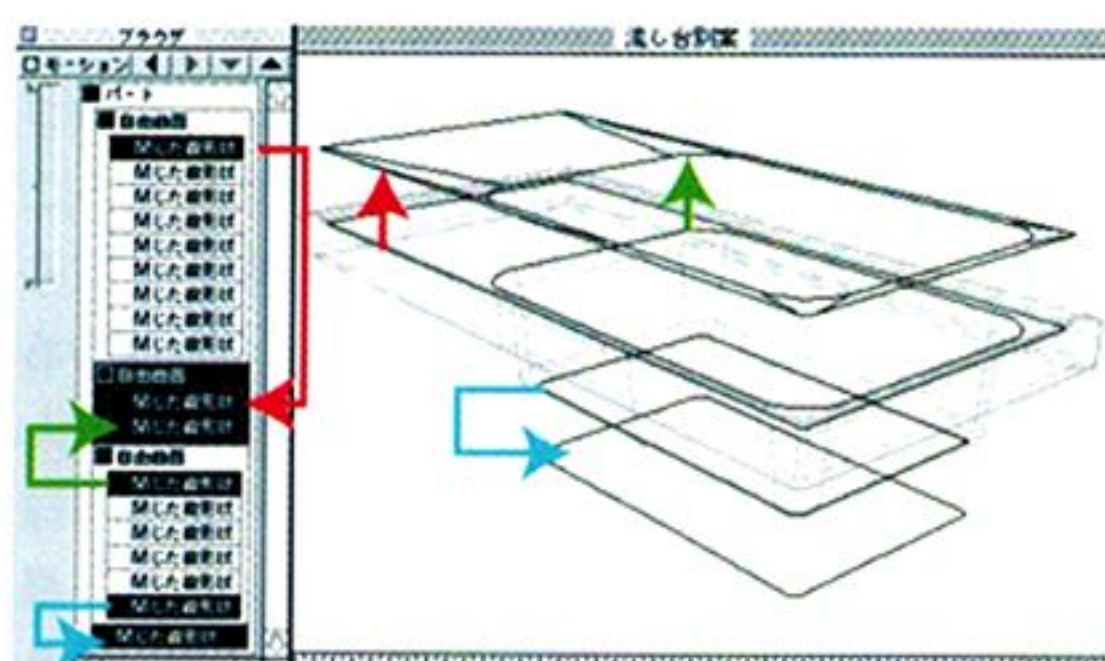



図23　掃引がどのような感じで行われているかを示す図。別パーツを組み合わせて構成されている

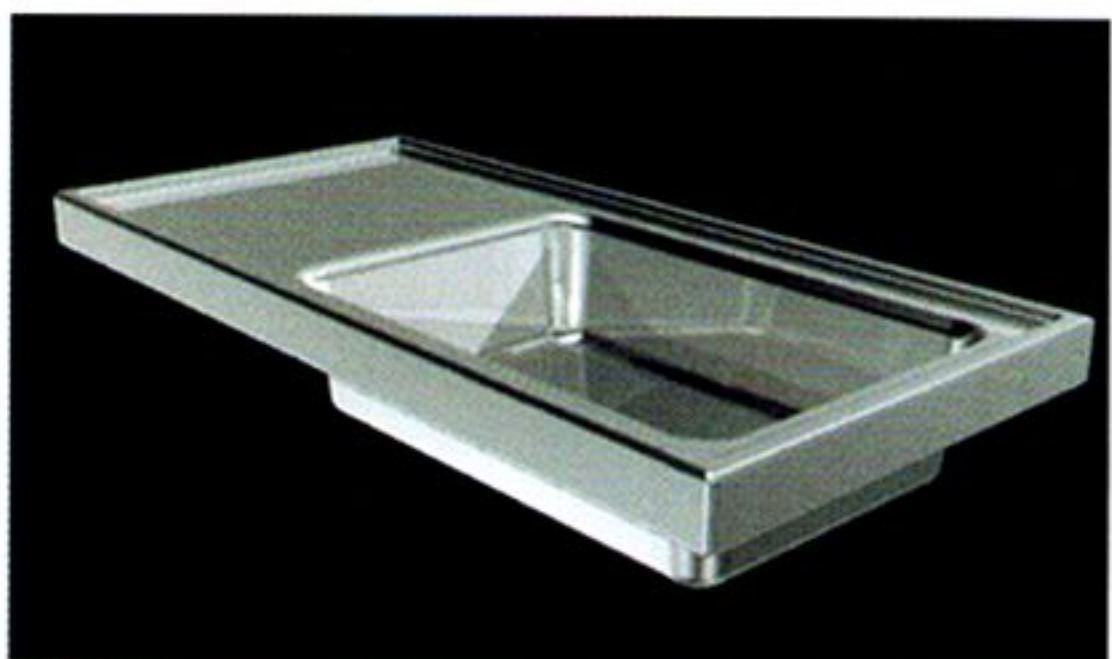



図24　同じくレンダリング画像。掃引方法が違ってもできるものはほとんど変わらない。使いやすいほうを選べばよいだろう

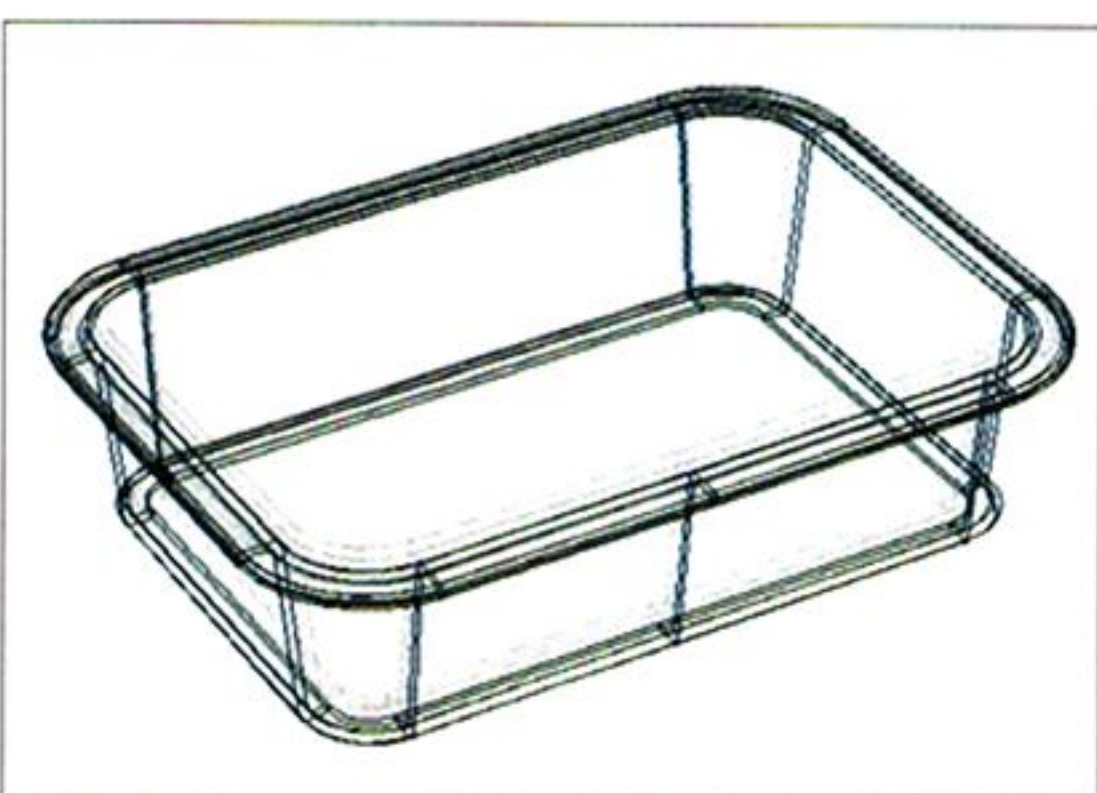



図25　同じように掃引を繰り返した食器入れの例

図26　レンダリングするとこんな感じ。掃引を上手く使うと効率がよさそうなのがわかるだろう

ですが，今回の絵では窓からの光や流しの上にあるであろう小さい照明を無視するわけにはいきません。人物の光の当たり方もその辺りを意識して描いています。

配置が終わったら最終レンダリングの前に人物と背景のパースをあわせるわけですが，まずすべきことは，水平線の高さをどの辺りに取るかを決め，人物と背景の水平線の高さを一致させることです。水平線の高さとはすなわち背景のパースの線が消失する点の高さであり，カメラの目線の高さです。カメラの目線の高さが人物の目の高さと同じなら，水平線の高さは人物の目の高さになります。

今回は人物とのパースをあわせるためにテンプレート機能（下絵取り込み機能）を使用しました。この絵の人物の場合，左右のお尻がほぼ真横に並んでいるため，お尻の辺りに水平線を設定しない限り，あまりきついパースはつけられません。もう少しカメラ位置を高くしたいのですが，非常に微妙なパースとなり，背景とあわせるにはテンプレート機能に頼らざるをえません。

まず下絵はモニタで見れるサイズに加工したものを別に用意します。下絵は全画面にひとつしか表示できないので，読み込んだあとはパースペク

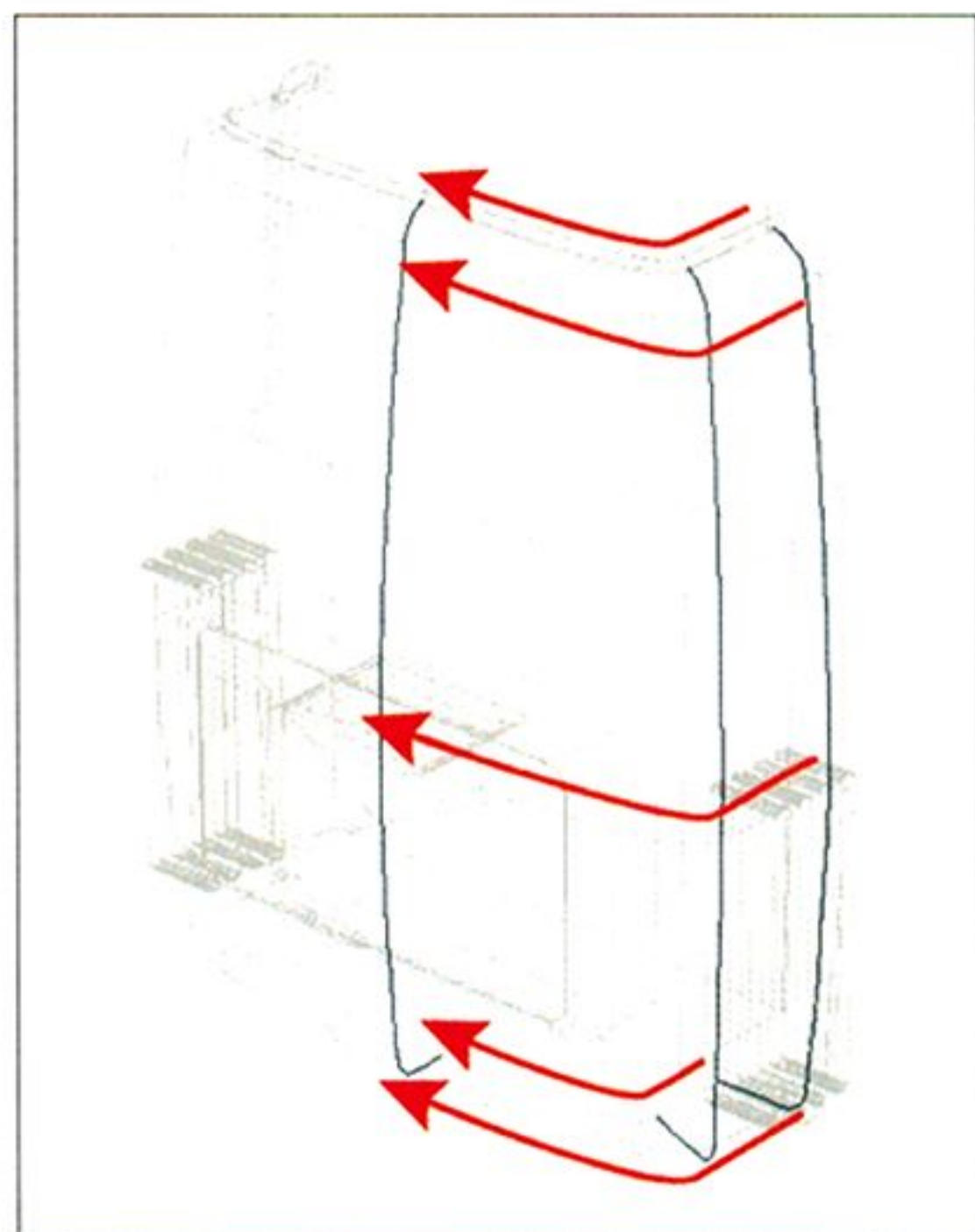



図27　微妙な曲面を持った湯沸かし器のカバーも掃引で作ってみよう

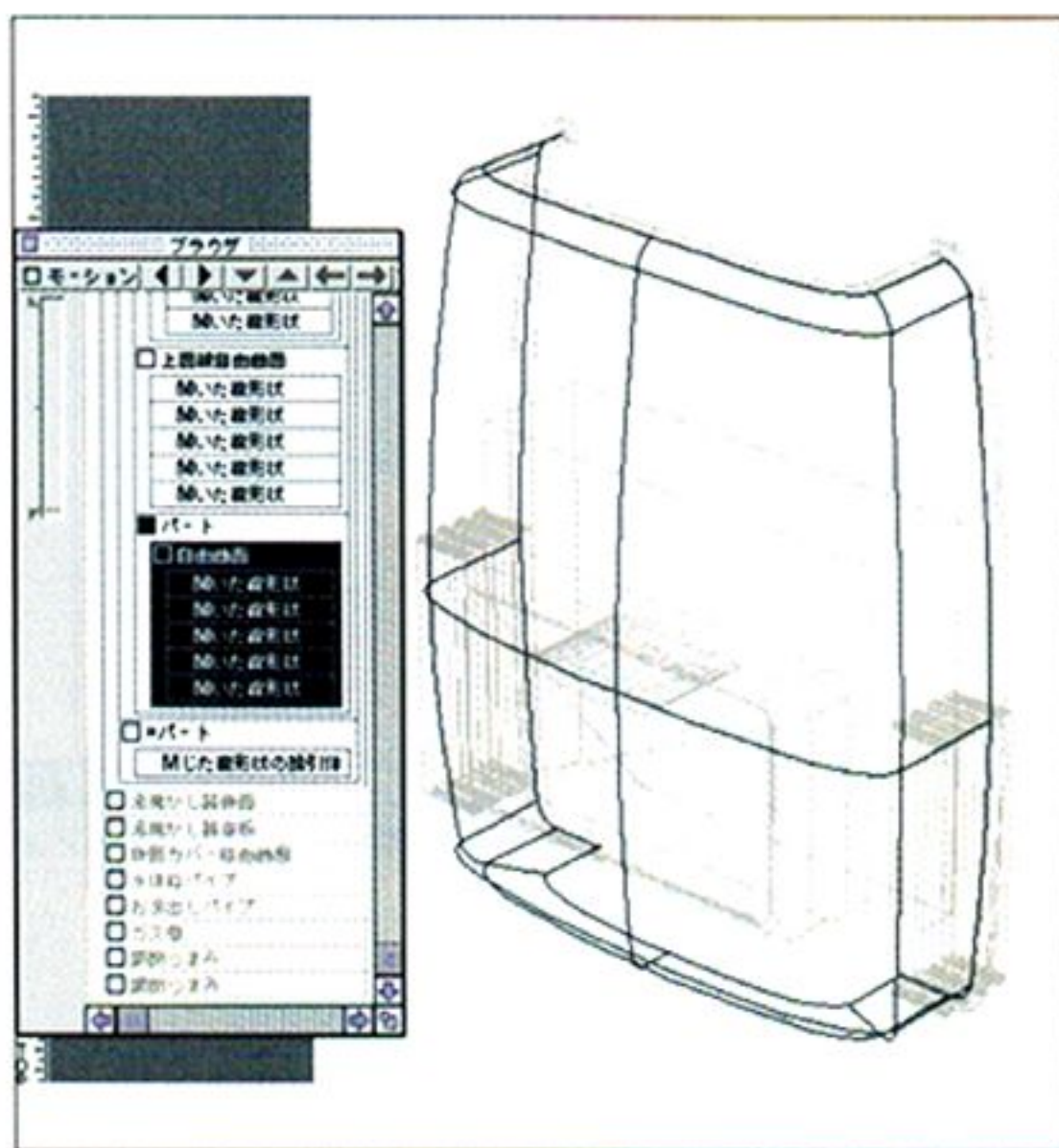



図28　左右対称なので片方を反転コピーして一体化させる

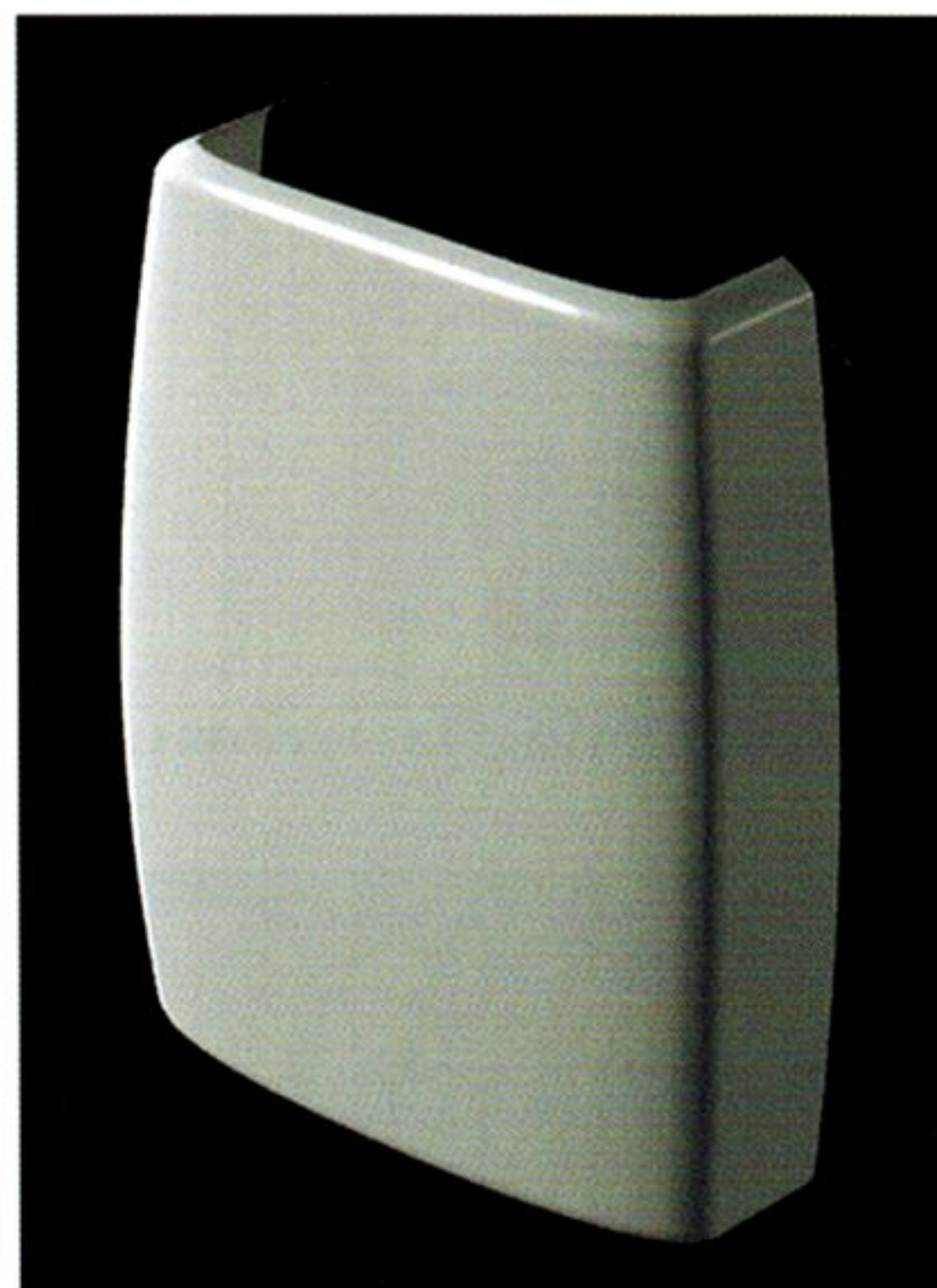



図29　レンダリングしてみたところ。ほかの方法で作るとなるとかなり大変そうな形状だ

図30　これはバンプマッピングで蛇腹の金属パイプを作成した例。十分に効果が出ている

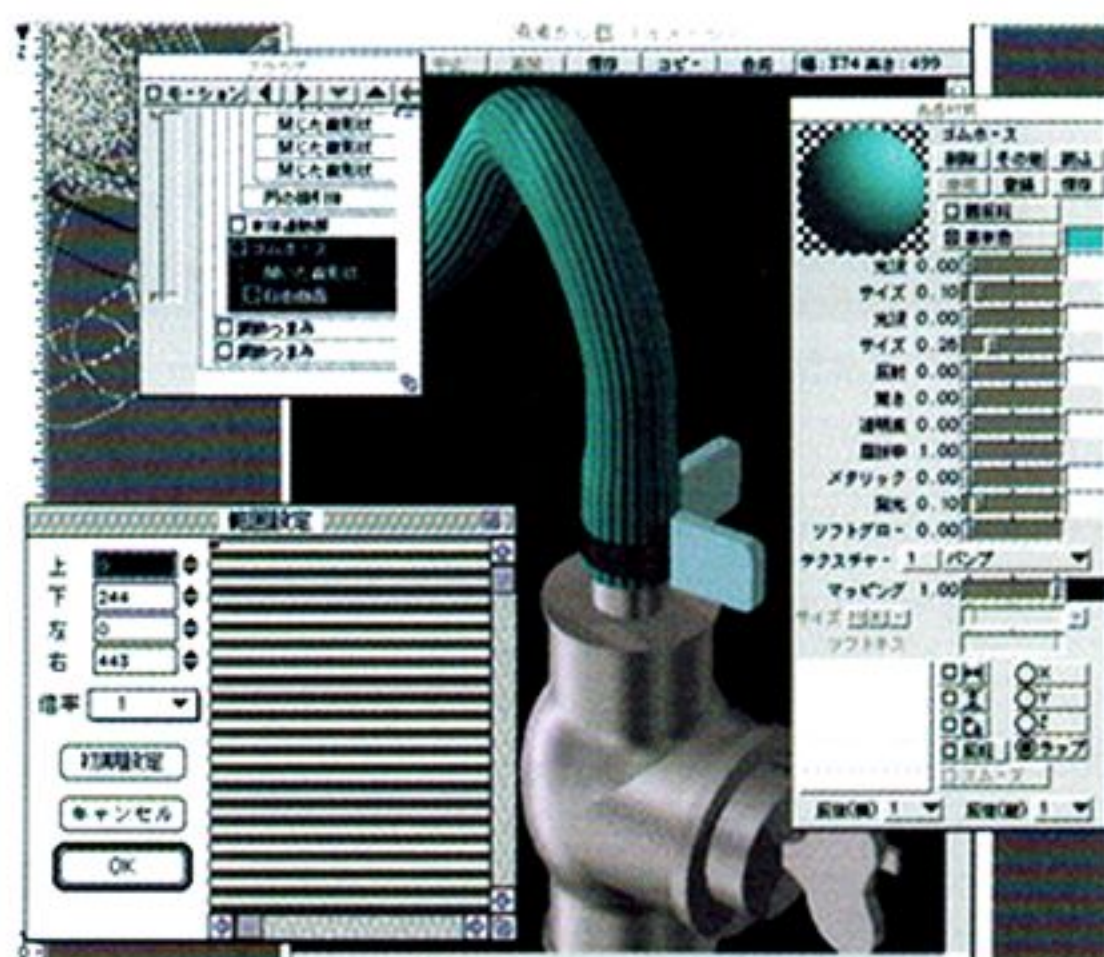



図31　同じバンプを横だけに使うとゴムホースがいかにもそれらしくできあがる

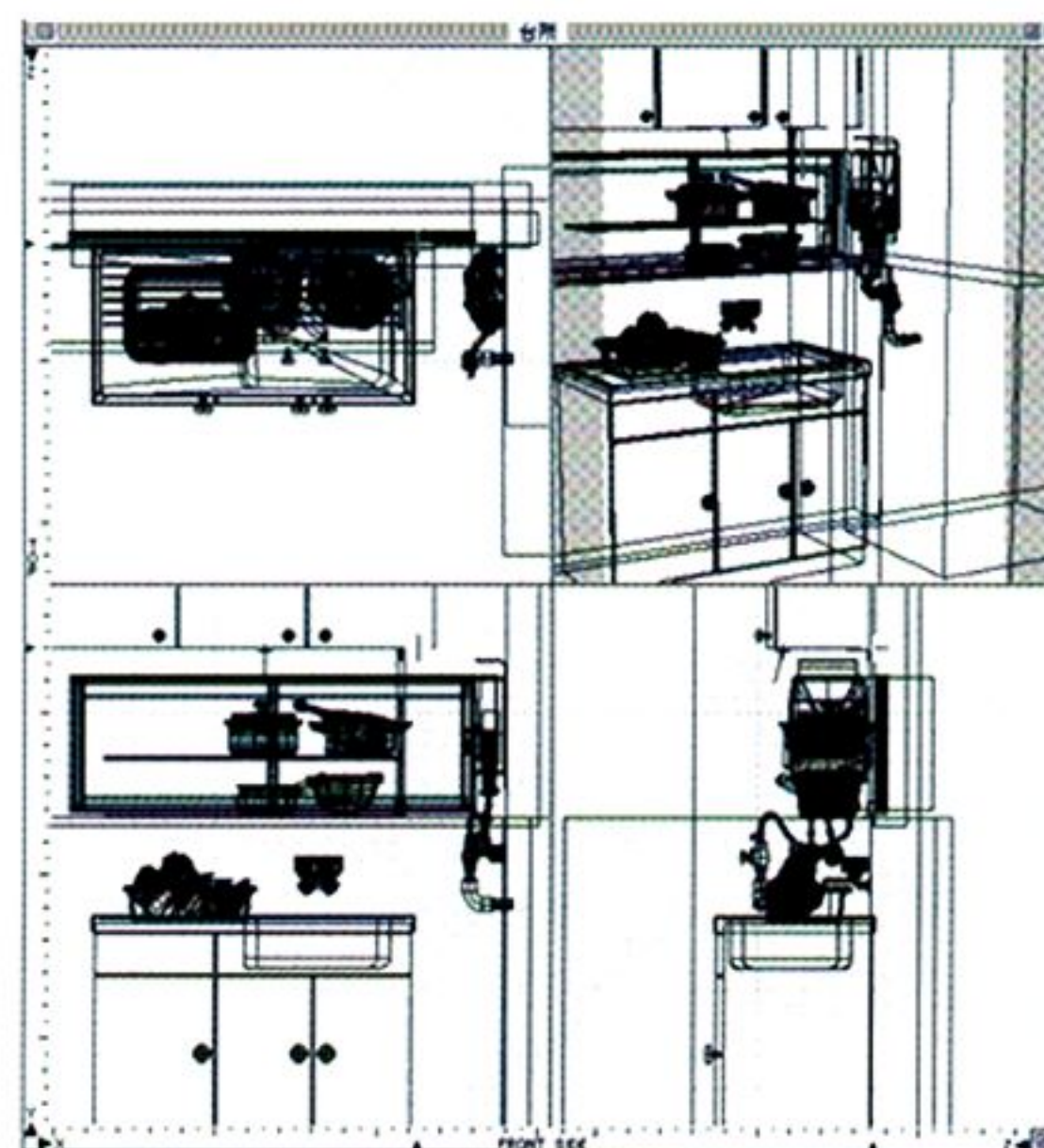



図32　これまでに作ったオブジェクトを配置してシーンセッティングを行う

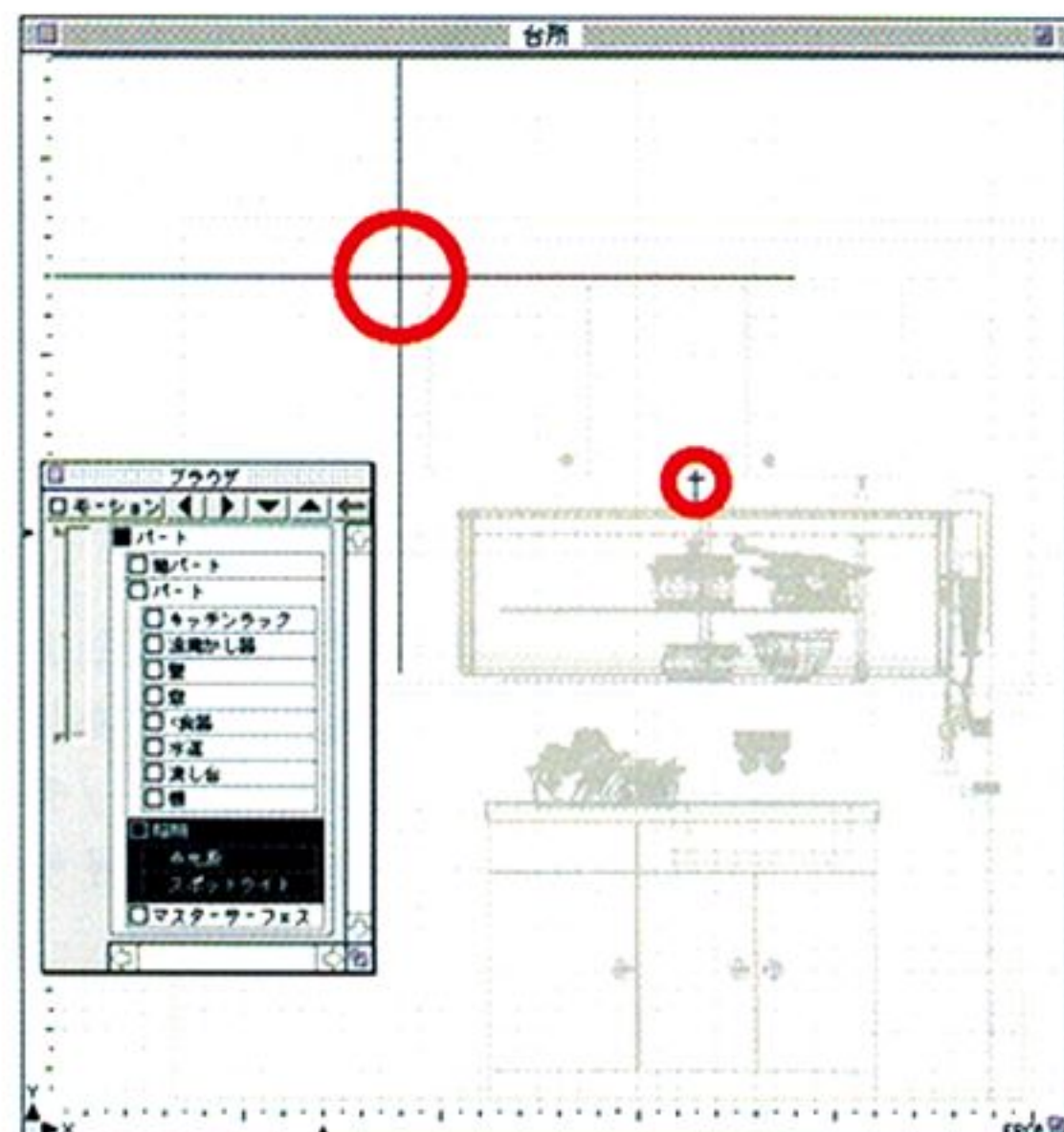



図33　光源を配置しよう。十字架上のマーク（赤いマーク付き）が光源だ

光源

明るさ	100.00000	
□スポットライト		
□スクエア		
角度	60.00000	
ソフトネス	0.00000	
影	0.44000	
影のソフトネス	0.58000	
光沢	1.00000	
拡散反射	1.00000	

キャンセル　OK

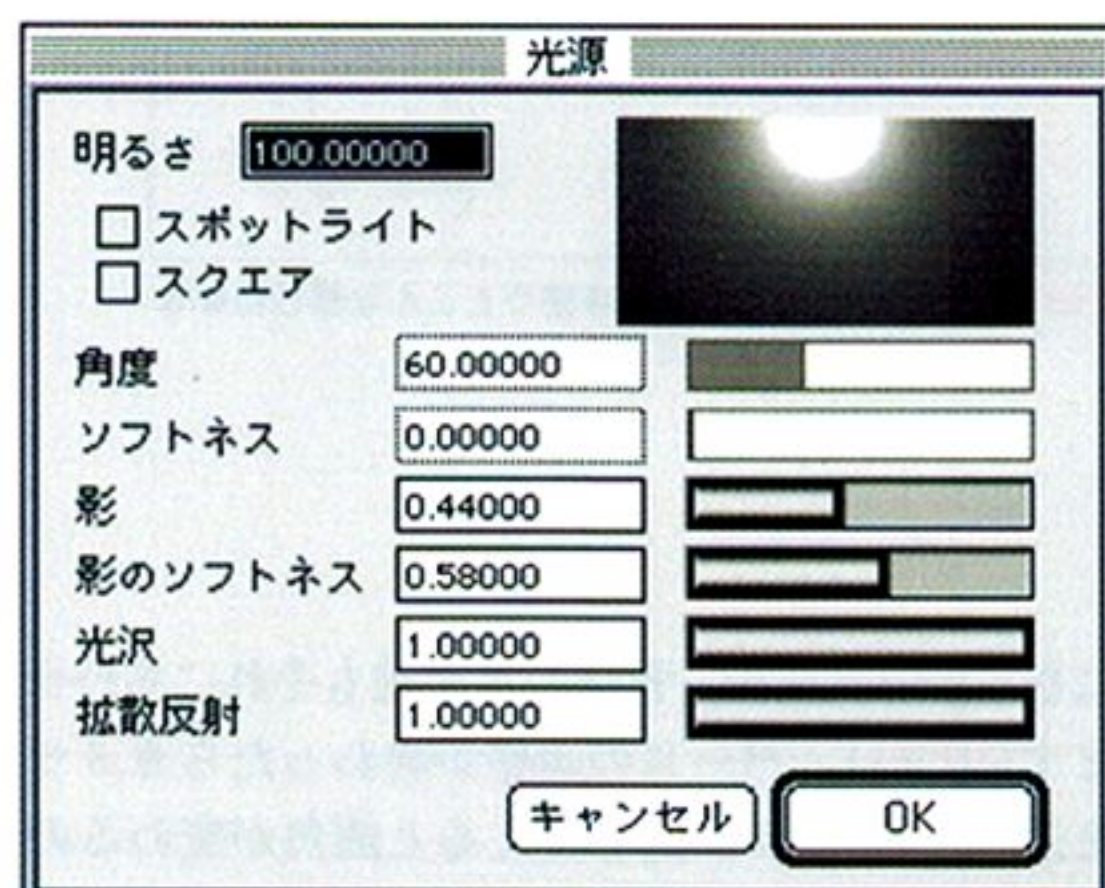



図34　光の種類をここでいろいろ設定できる。室内灯っぽい設定を作る

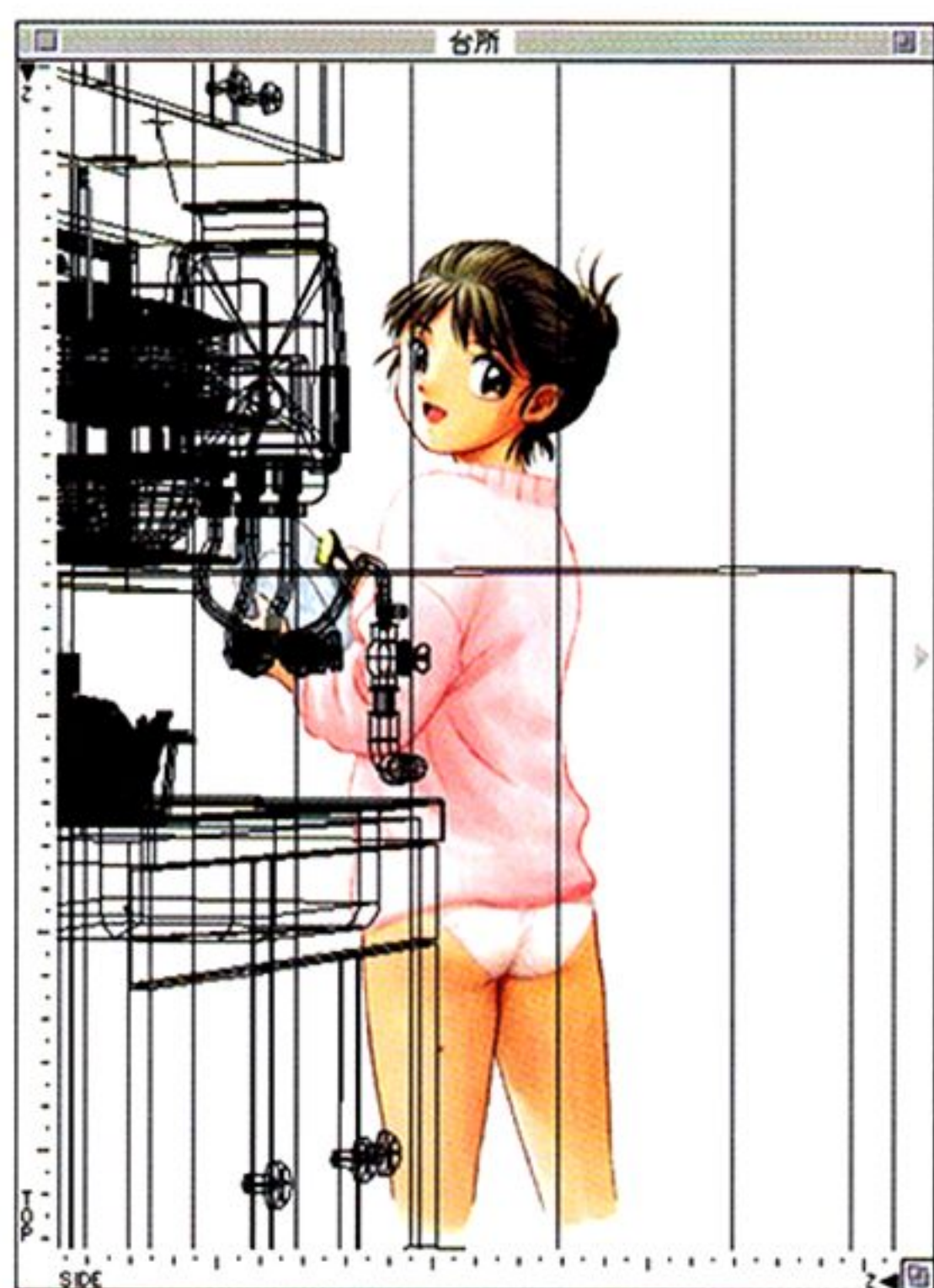



図35　テンプレート機能を使って下絵との合成の調整をしていく

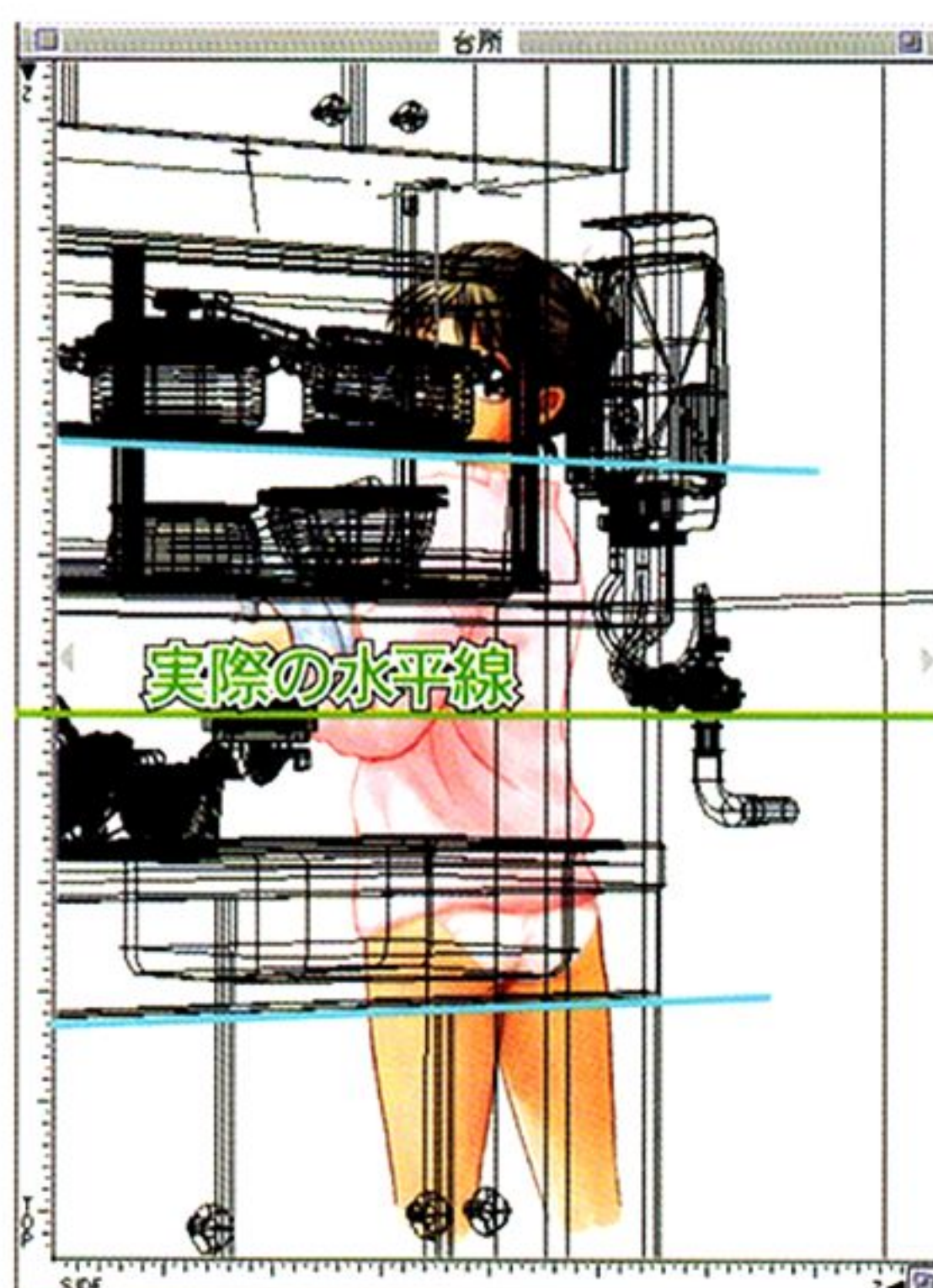



図36　キャラクターに対する水平線と合致するようにオブジェクトを配置していこう

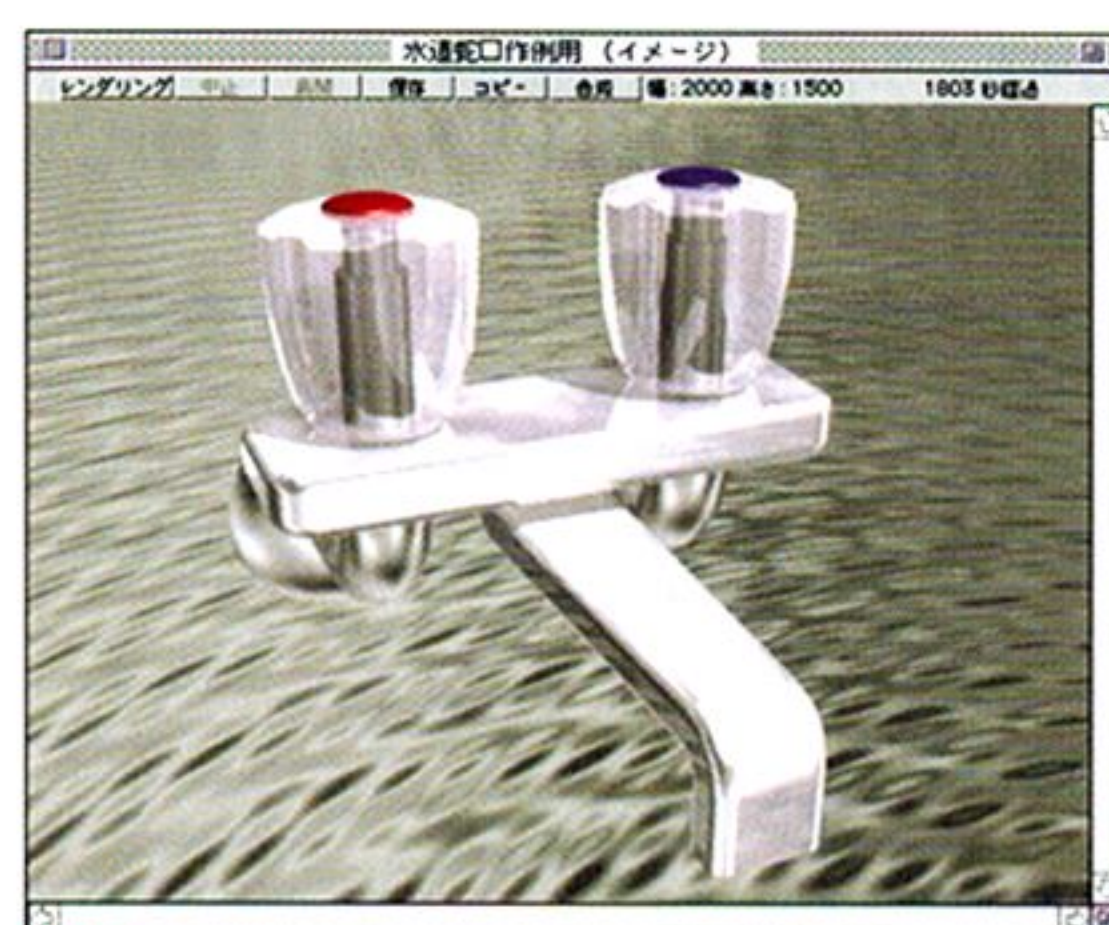



図37　イラスト調にする前のレイトレーシング画像。いかにもフォトリアリスティック

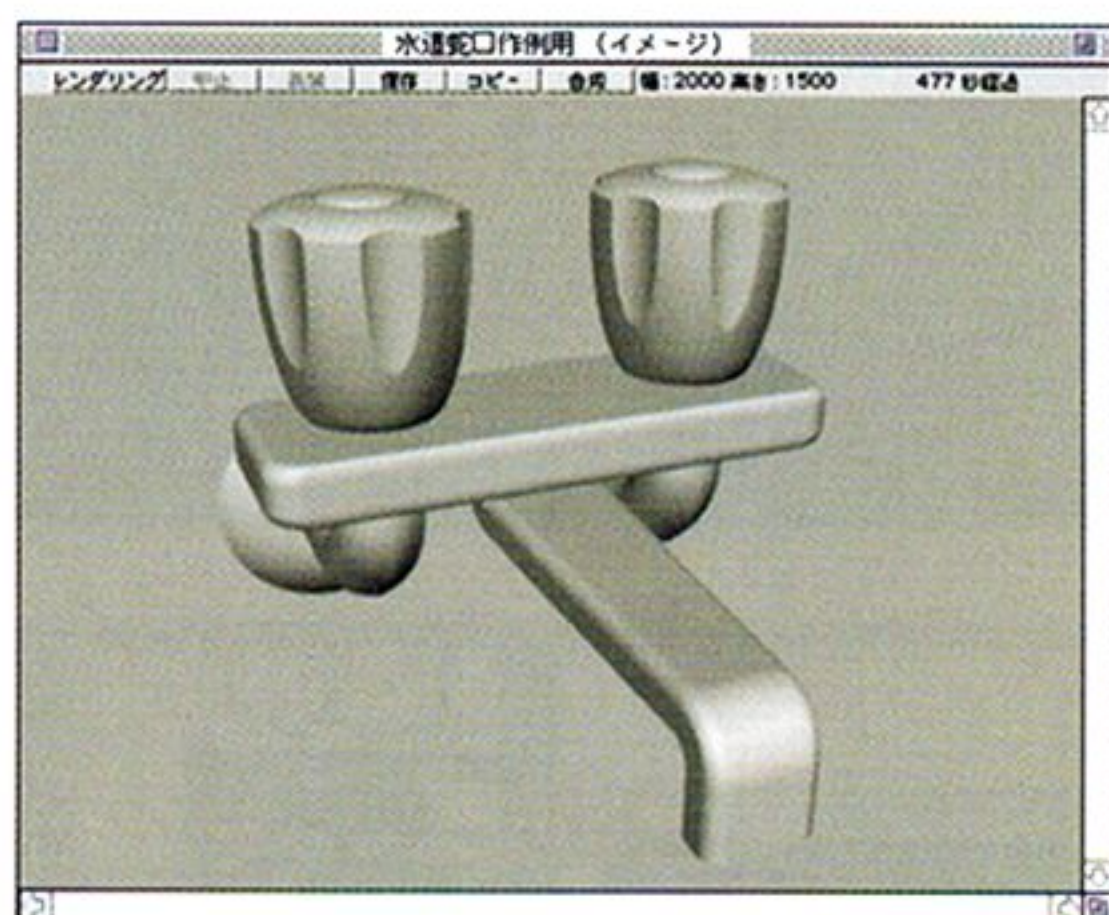



図38　次に材質設定などを落としたものを同様にレンダリングしておく

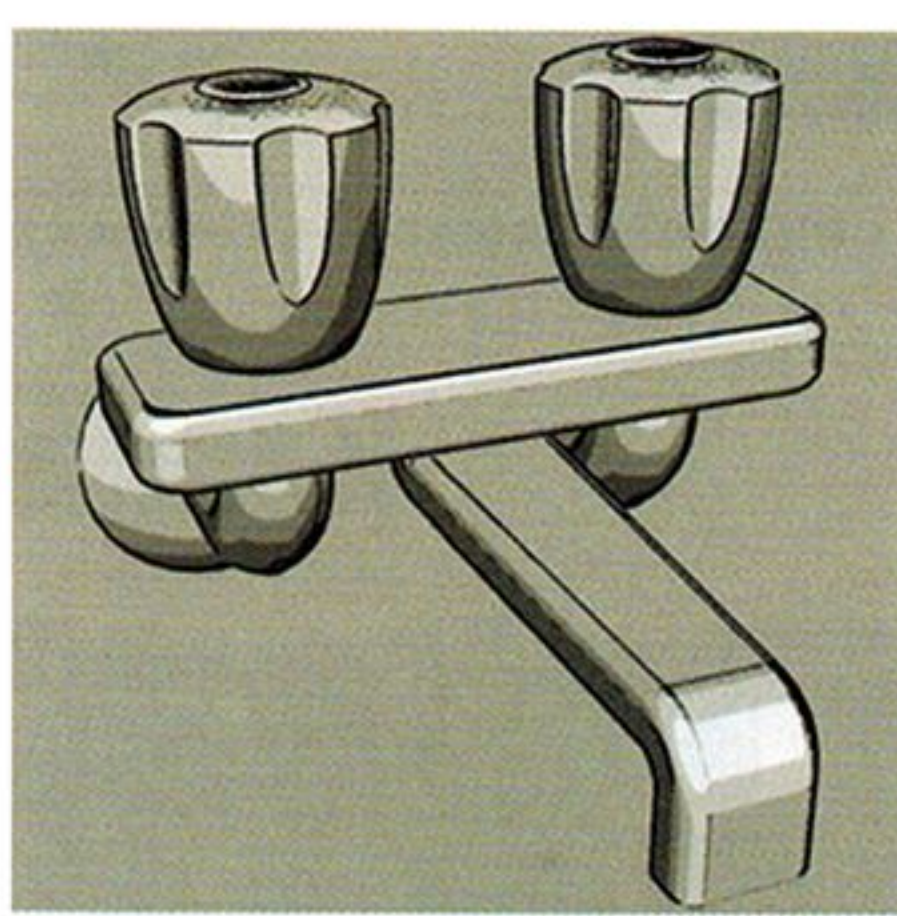



図40　エッジのポスタリゼーションを行って……

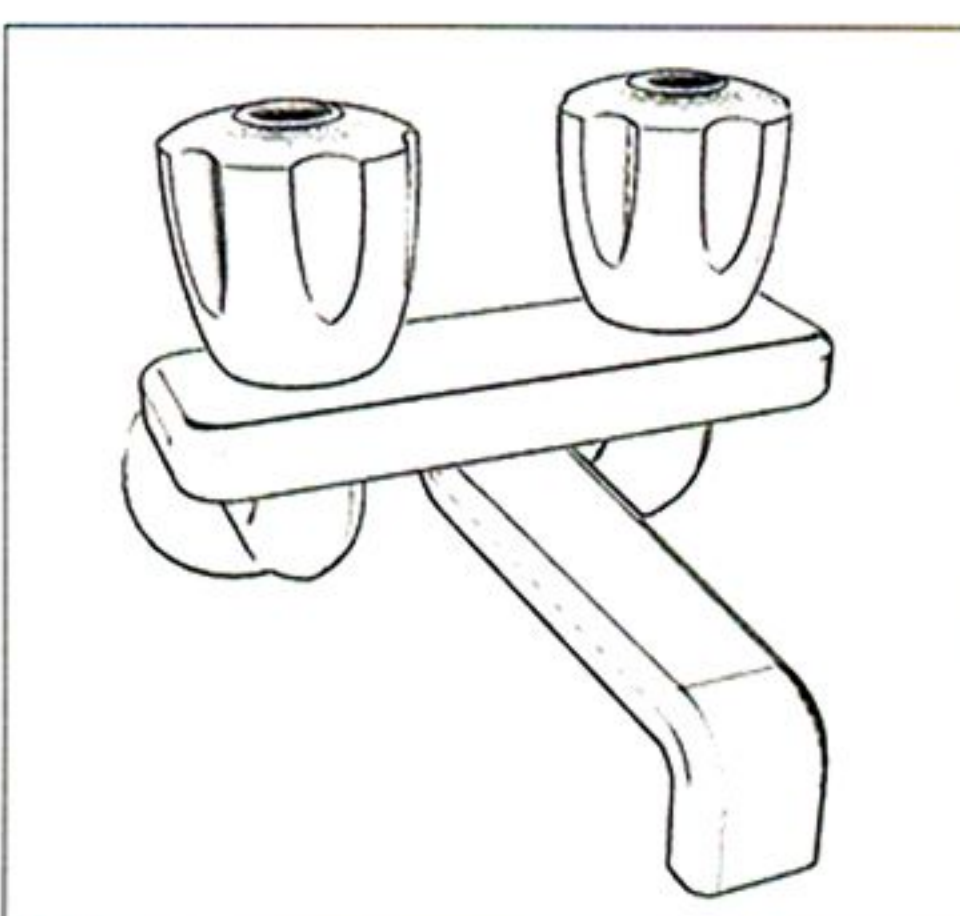



図41　2値化を行うと輪郭が抽出される

図39　レイトレ画像にドライブラシをかけたところ

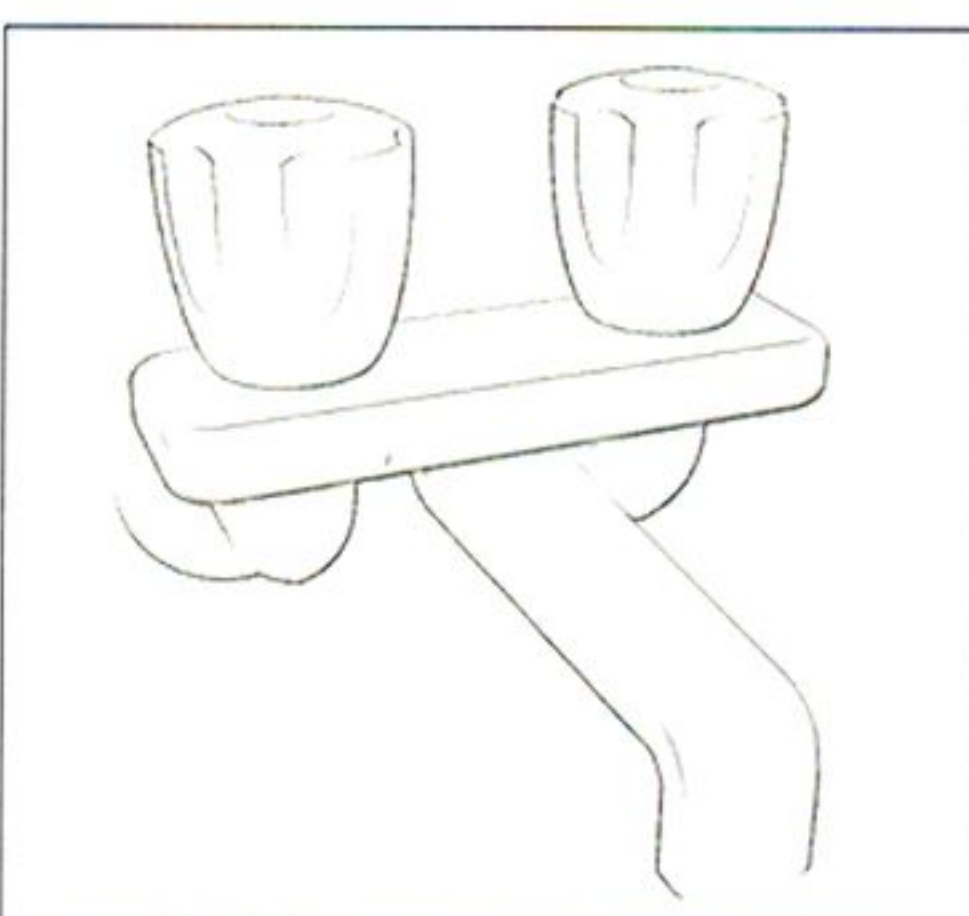



図42　輪郭抽出機能をそのまま使うとこんな感じになる

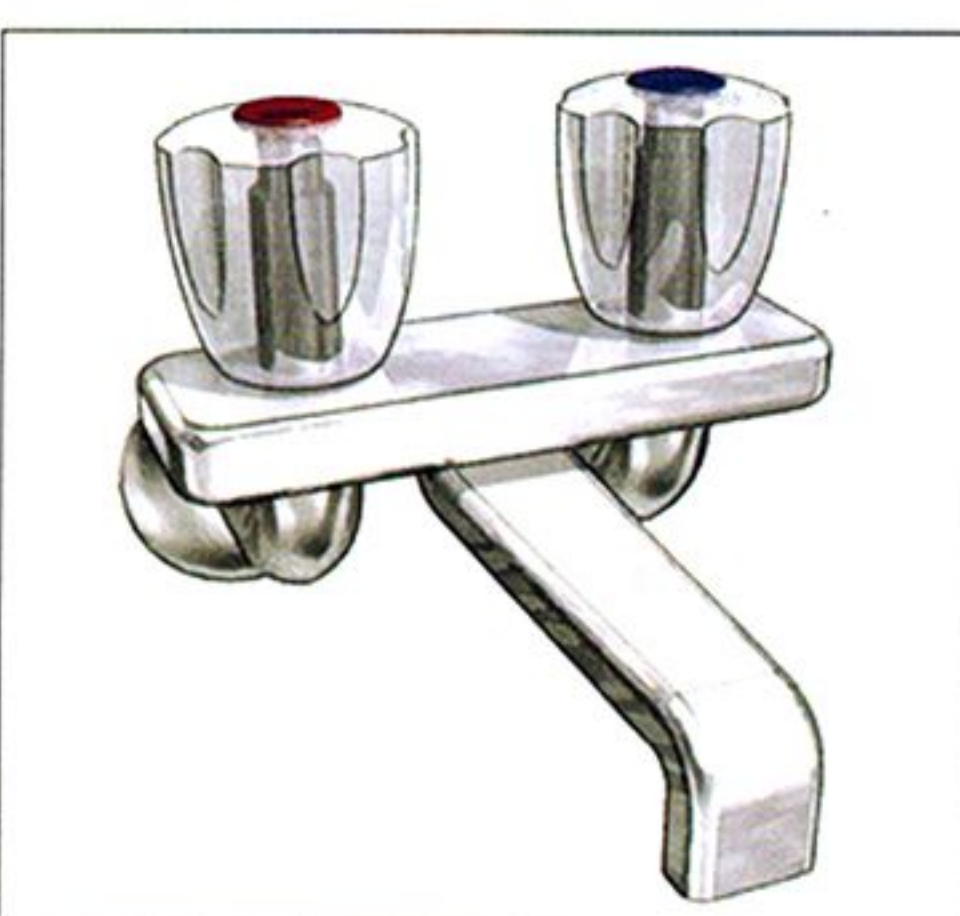



図43　レンダリング画像と主線抽出された画像を合成することでイラスト風の仕上がりにできる

ティブ画面のみを表示させます(図35)。「テンプレートサイズ」をチェックしておけば厳密なサイズで表示できます(前回はこの機能に気づいてませんでした。勉強不足ですいません。R2以前も下絵取り込み機能は十分使えます)。あとは人物との大きさの対比を考えてズームを調整し，回転したあとに位置を調整。人物の水平線がどの高さになるかを見極め，背景の水平線もそれにあわせます(図36)。パースの調整が終わったらカメラを退いて(カメラ位置を変えると画角が変わるので実際には望遠で調節)，背景が多少余分にレンダリングされるようにしておきます。

今回はフォトリアルなレンダリング画像を手描き風に仕上げるのが目的です。フォトリアルな画像に対して手描きイラストの特徴とはなんでしょうか。手描きのイラストといってもいろいろありますが，この場合はマンガ調の絵に馴染むタッチのイラスト，すなわち主線(おもせん)のあるイラストが求められます。

極端ないい方をするなら，フォトリアルな画像にそのまま主線を重ねただけでも随分と手描きの

図44　今回使用する背景のレイトレースレンダリング画像

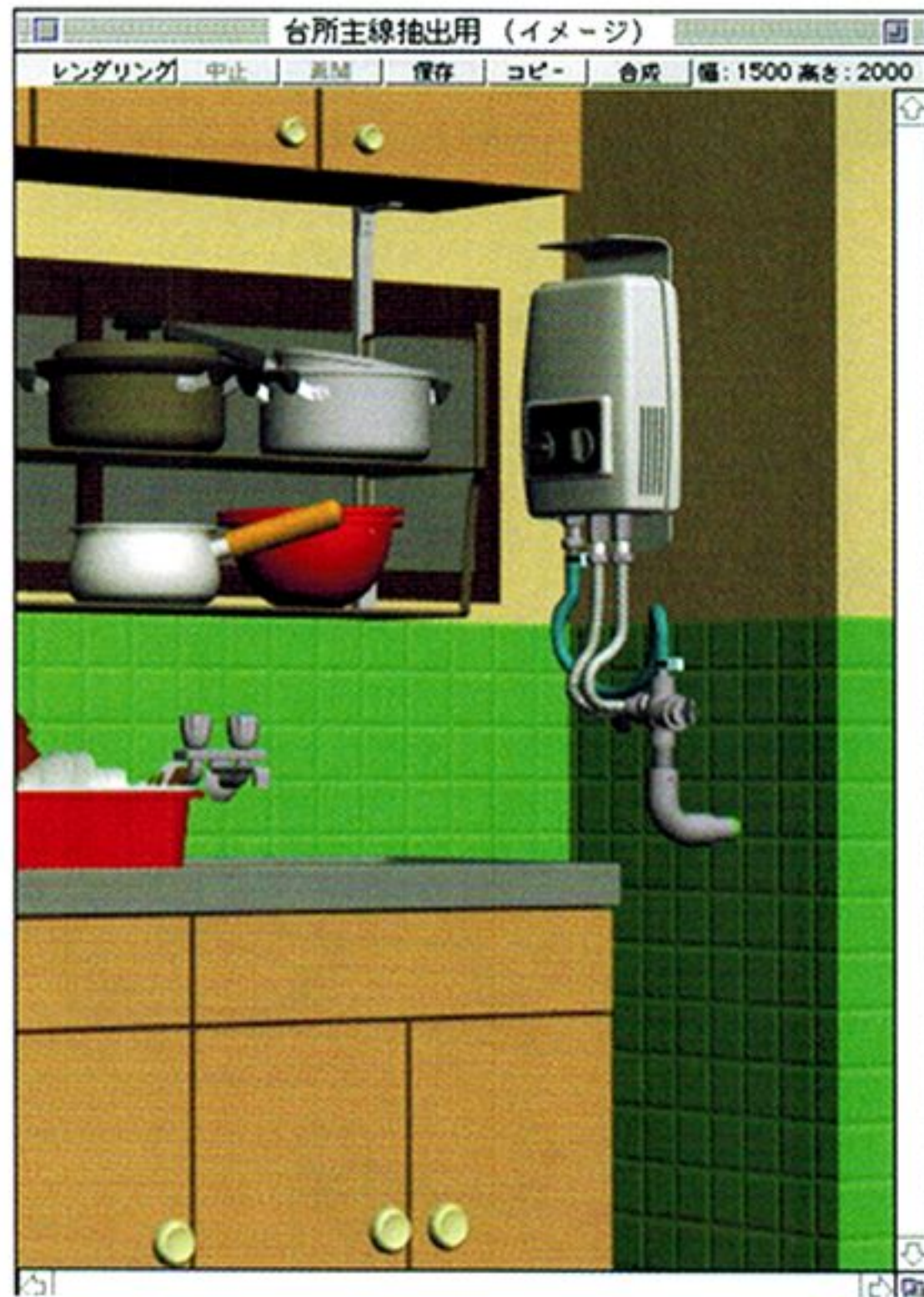



図45　こちらは同じく，主線抽出のためのレンダリング画像。主線の品質が重要なので解像度は高めにしたい

図46　スムーズシェーディング指定だと陰影がエッジの太さになってメリハリがつく

図47　ハイライトがあると誤動作の原因になるので主線抽出用のレンダリングでは光沢設定は厳禁だ

感じに近くなります。ならばあとはそのフォトリアル画像に「アーティスティック」に属するフィルタをかけることで，イラスト調に仕上げることができるはずです。

つまりフォトリアルな画像をイラスト化するには，通常のレンダリング画像のほかに主線抽出用の画像，すなわち「まったく同じ画像から極端なてかりや余計な表面属性を排除し，影を落とさないようにするなど，不必要なエッジを拾わないように加工した画像」が必要になります。通常の画像と，影やてかりのない画像を2つ用意するという芸当は普通の写真ではできません。3Dソフトならではの技法ではないかと思います。ついでにいうなら，オブジェクトごとにレンダリングするとオブジェクトのマスクも作ってくれるので，個別にレンダリングしてPhotoshop上で合成するのも比較的容易です。

イラスト化の手順はレンダリング時間のことさえ考えなければ非常に単純です。まずモデリングしたものを普通にレイトレーシングでレンダリングして画像を保存します(図37)。そしてオブジェクトの表面材質をはがして「影を落とさない」「背景を反映しない」設定で同様にレンダリングします(図38)。できあがった2枚の画像をPhotoshopに読み込んでそれぞれフィルタをかけます。通常の画像には「アーティスティック：ドライブラシ」がいいでしょう(図39)。

主線抽出の方法は2とおりあります。ひとつは主線抽出用画像に「アーティスティック：エッジのポスタリゼーション」をかけたあと(図40)，二値化して黒線のみを残す方法(図41)。もうひとつは「表現手法：輪郭検出」をかける方法です(図42)。主線を抽出すれば，あとはドライブラシをかけた画像に主線画像を「乗算」で重ねるだけです。私の場合は「エッジのポスタリゼーション」で得た主線を透明度40%，「輪郭検出」で得た主線を100%で重ねています(図43)。

ただし，「食器」「流し台」といった1個1個のオブジェクトならともかく，今回のようにひとまとまりの背景をイラスト化する場合，主線の太さをどう取るか……，どれくらい粗いタッチにするかなどをある程度考えてかからないと，主線が細すぎて，あってもなくても同じとか，逆に太すぎて絵が粗くなり，手で描いたほうがマシだったなどということになりかねません。

実際に「塗り」の部分を担う画像，すなわち通常のレンダリング画像は図44のようになります。この画像はエッジはそれほどシャープでなくてもいいので，レンダリング時間を節約するため可能な限り小さいサイズでレンダリングしておいて，主線との合成の段階でサイズを調整するのがよいでしょう。どうしても時間がもったいないという人は，スキャンラインで「材質を表示」をチェックしてレンダリングし，影をあとから描き加えるという手もあります。

気をつかうのは，むしろ図45の主線抽出用の画像です。この画像にはエッジの滑らかさが求められるため，それなりのサイズでレンダリングする必要があります。時間節約のためスキャンラインでレンダリングしますが，フラットシェーディングだと陰の境界線を拾うので「材質を表示」をチェックして滑らかにレンダリングされるようにします。

滑らかな陰は，Photoshop上でエッジを抽出したときに線の太さという形で反映されるので単調な太さの線を拾うよりは都合がいいのです(例：図46)。ライトもできれば1灯にとどめておいたほうがいいでしょう。私は無限遠光源をほぼ正面の若干斜め上から当てています。影の方向についてはあまり気にする必要はないと思います。あとは表面材質設定を削除して主線を抽出しやすいように加工するのですが，注意すべきことは主線抽出の都合上，表面材質の「色」の設定だけは残しておいたほうがいいということと，湯沸かし器のパイプなど表面の凸凹を表すためのバンプマッピングはオブジェクトの「形」と見なすことができますから，ほかの設定を0にしてそれだけは残しておいたほうがいいということです。

あとは光沢などがあると，そのエッジを拾ってしまいますから，全オブジェクトを含むパートを選択し，新規の表面材質を設定して光沢など一切の設定を0にしておけば(図47)，色以外の表面材質を削除した全オブジェクトにその設定が継承され，光沢もなにもない絵がレンダリングされます。R3では光源のほうに光沢の設定がついていますが，それをうまく利用できないかどうか現在模索中です。

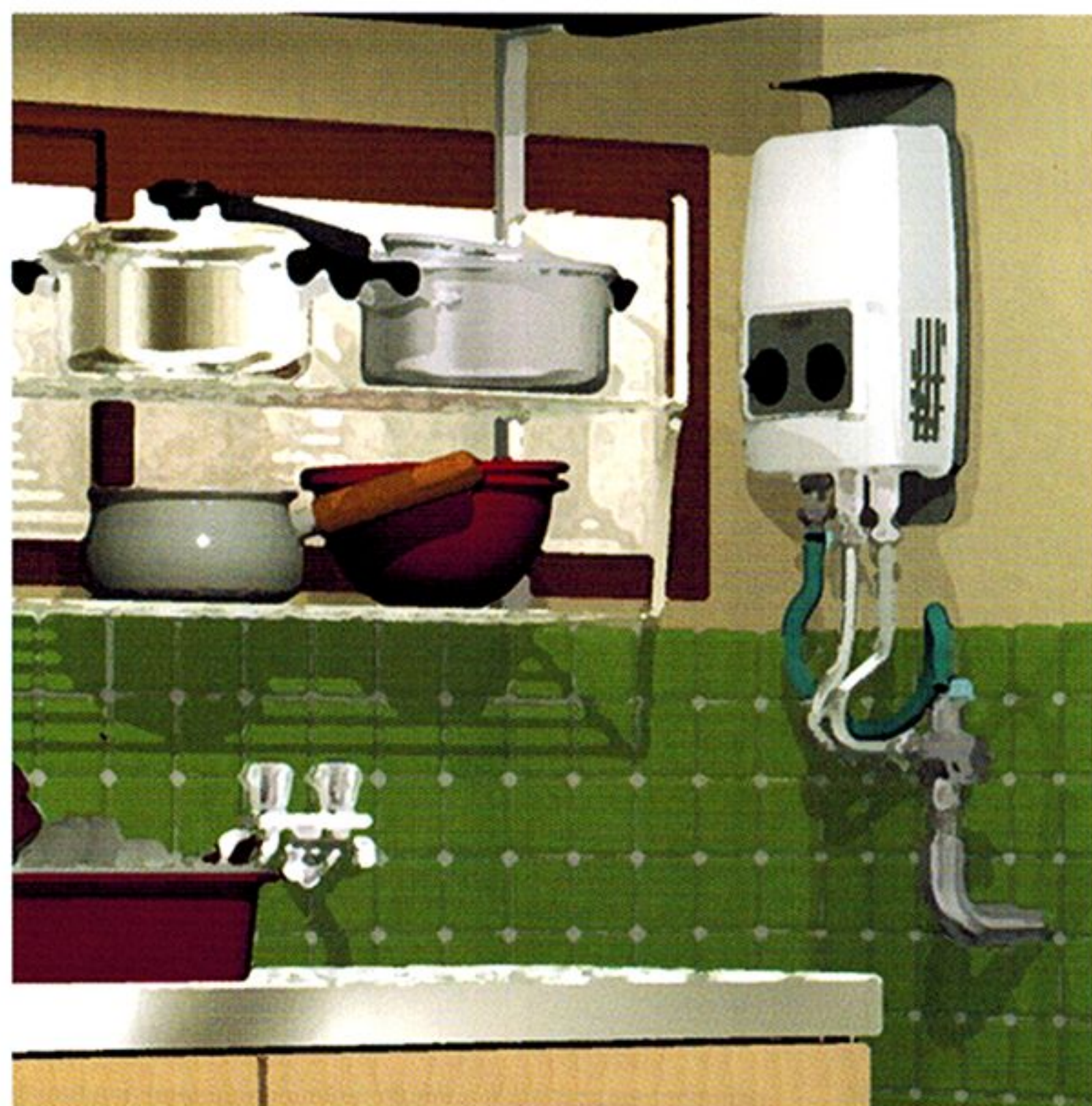



図48　前の例と同様に，色情報となるレンダリング画像にドライブラシをかけてリアル感を落としておく

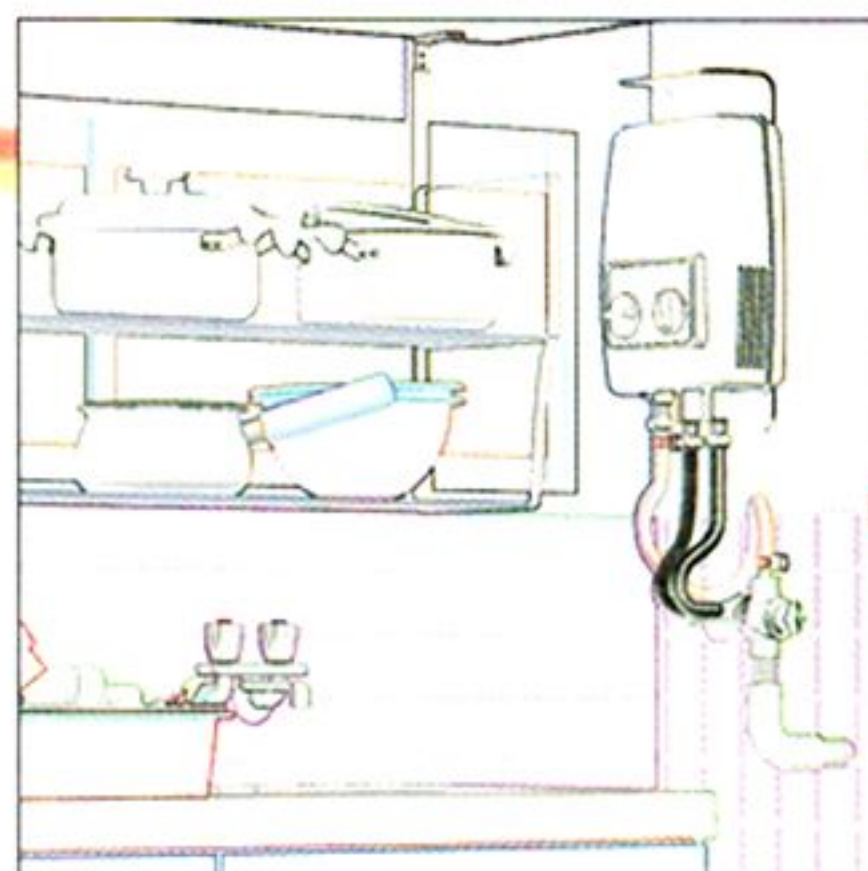



図49　主線用の画像に輪郭抽出フィルタをかけたところ

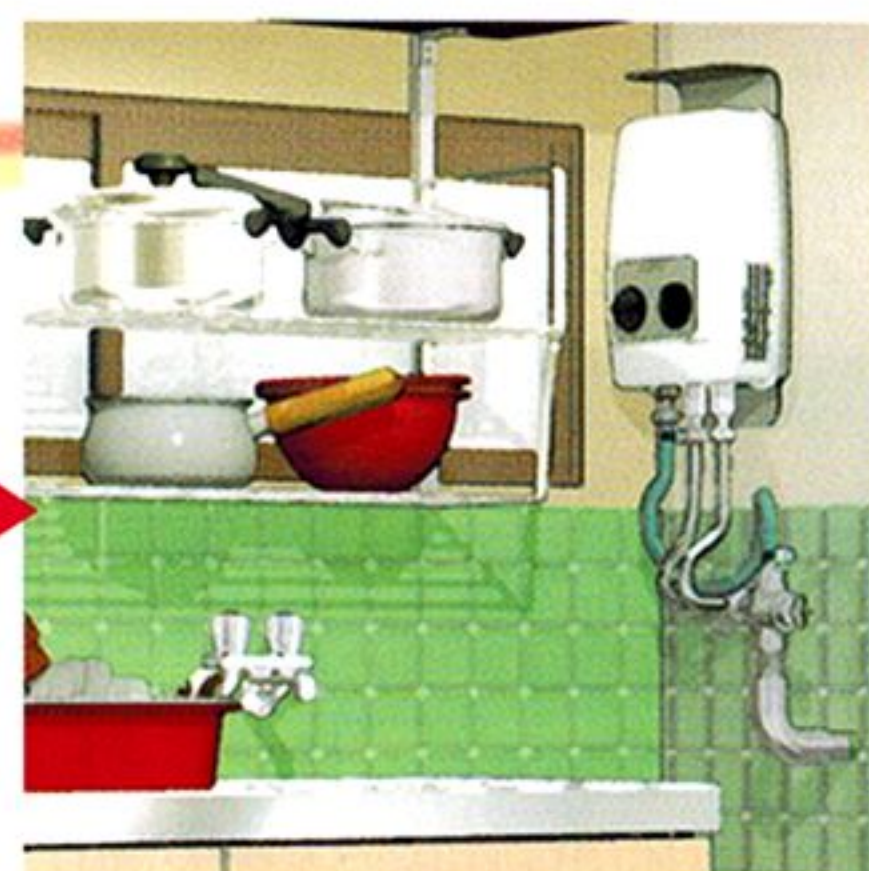



図50　輪郭抽出で得たエッジを塗りの画像と合成して仕上げた例

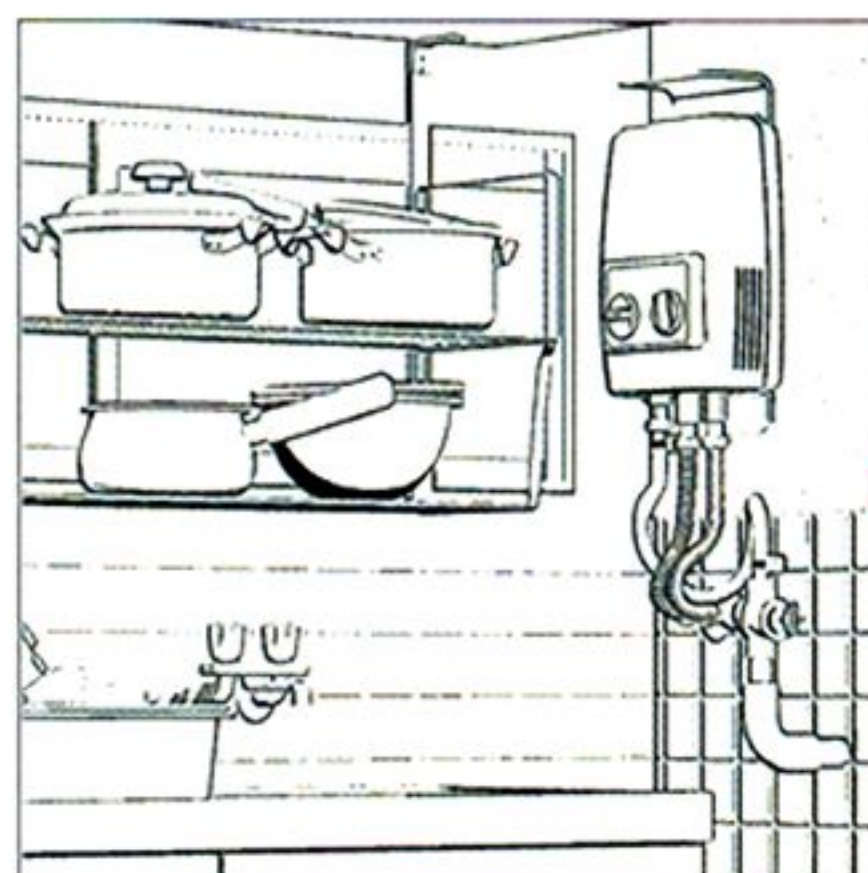



図51　こっちはエッジのポスタリゼーションと2値化で得た主線だ

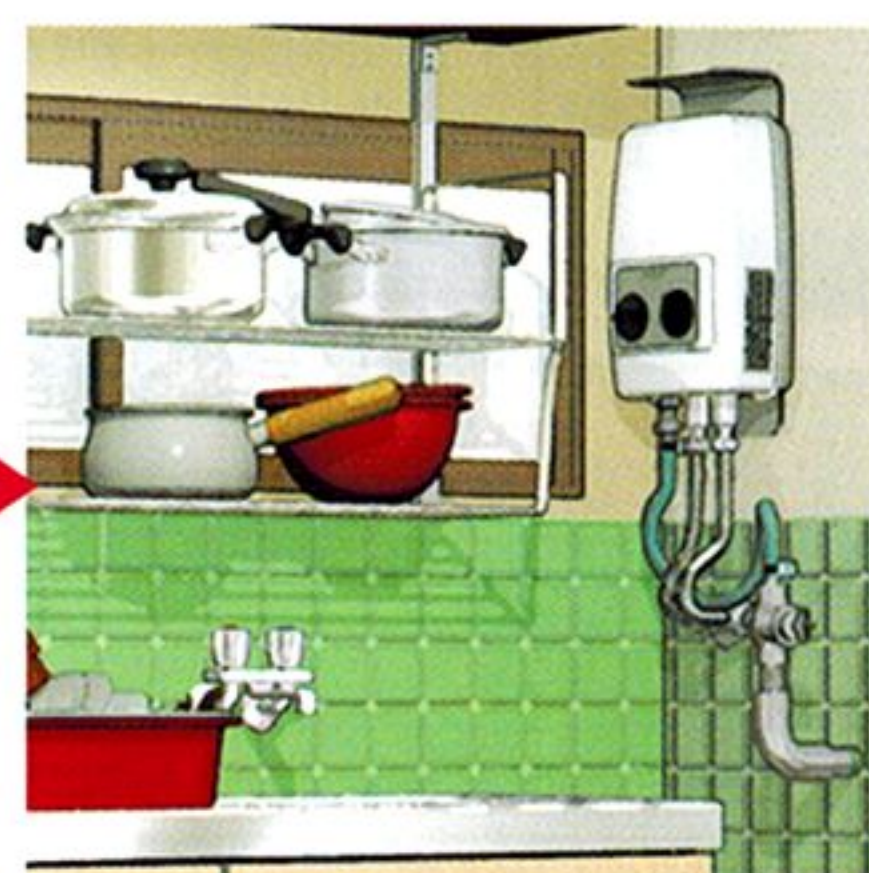



図52　2値化の主線と塗りを合成するとこんな感じになった

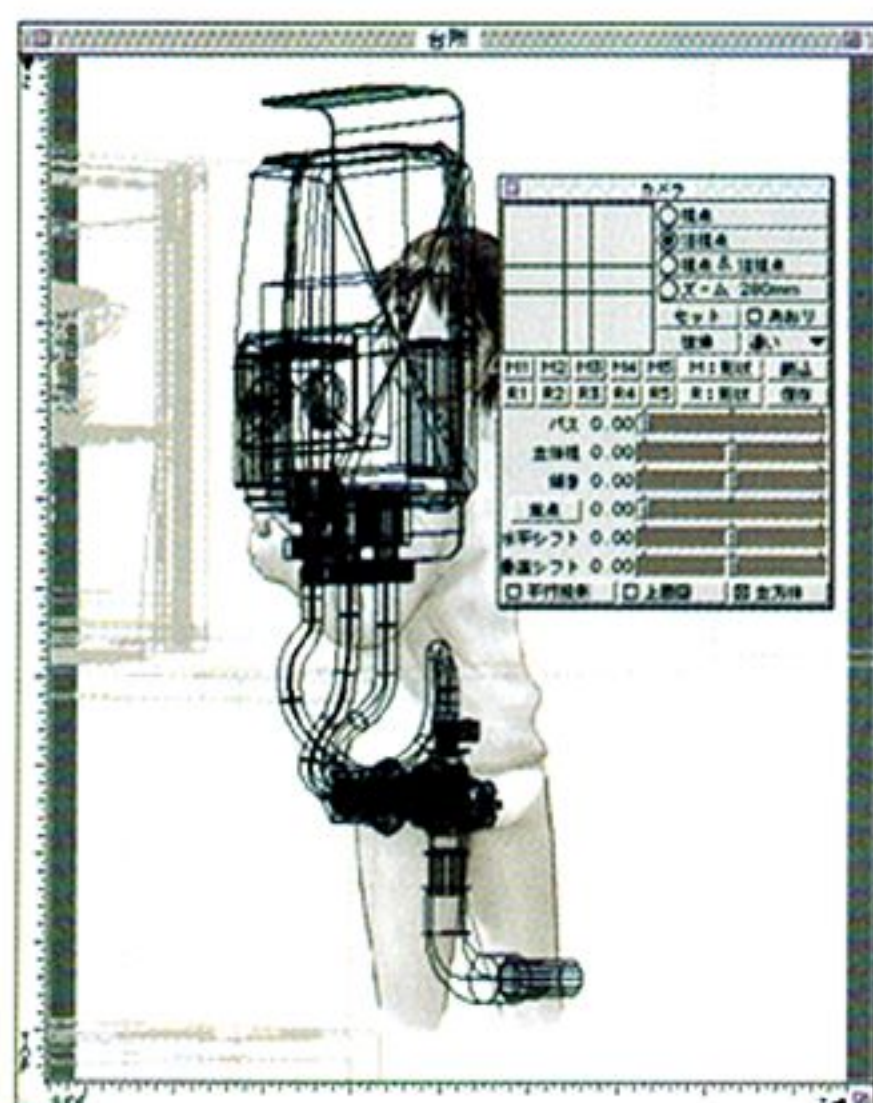



図53　メリハリをつけるため，要所要所でレンダリングし直して線の太さを変えてみよう

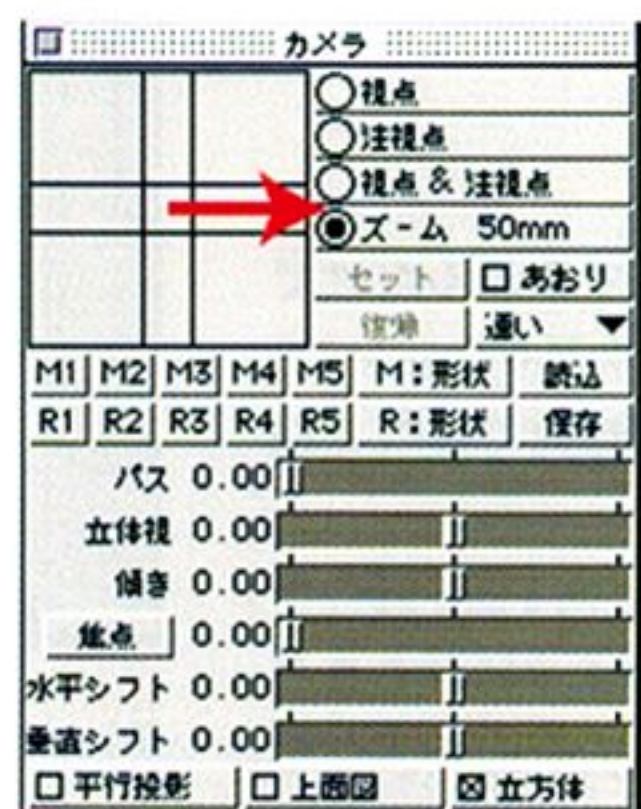
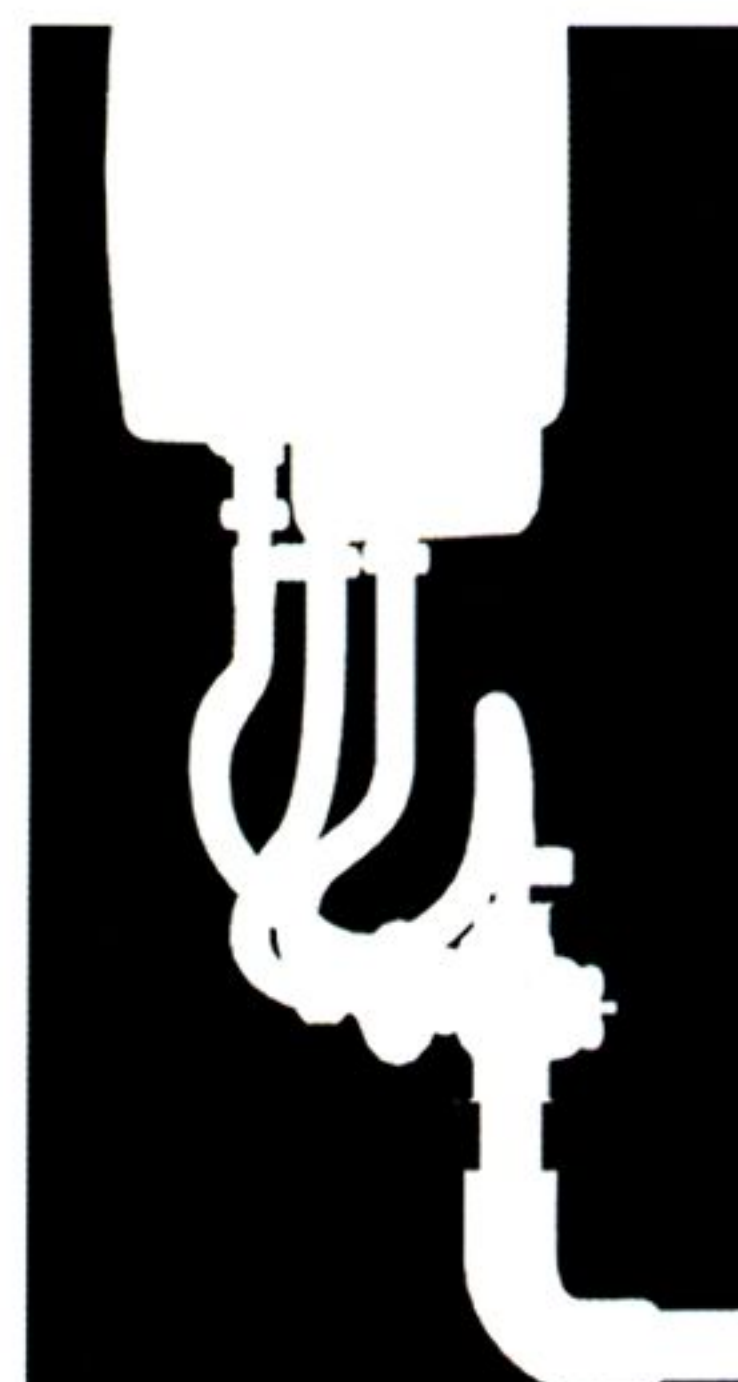



図54　画角が変わると合成できないので，ズームの際にはOptionキーを押して画角を変えない設定でズーム

3　背景の仕上げ

できあがった図44と図45のレンダリング画像を前述のイラスト化の手順に従ってPhotoshopで仕上げます。「塗り」の部分を担う図44の画像には「ドライブラシ」をかけます(図48)。設定値は画像の大きさによっても変わりますから一概にこうとはいえません。この画像は表現を担う部分でもありますから，ほかのフィルタもいろいろ試してみるといいと思います。次に図45の画像から主線を抽出します。まずレイヤーのコピーをとって前述の方法で2とおりの主線を抽出します。「輪郭検出」で得た画像は図49で，これを図48の画像に被せたのが図50です。ここでは合成方法に「乗算」ではなく，「ソフトライト」を選択しました。透明度は100%。これに「エッジのポスタリゼーション」→「二値化」で得た主線画像図51を「乗算・透明度40%」で被せたのが図52です。

これで一応完成としてもいいのですが，なんとなく不満が残ります。全体が同じような太さの線

図55-a　ディテールアップした湯沸かし器

図55-b　および，そのマスク画像(合成用)

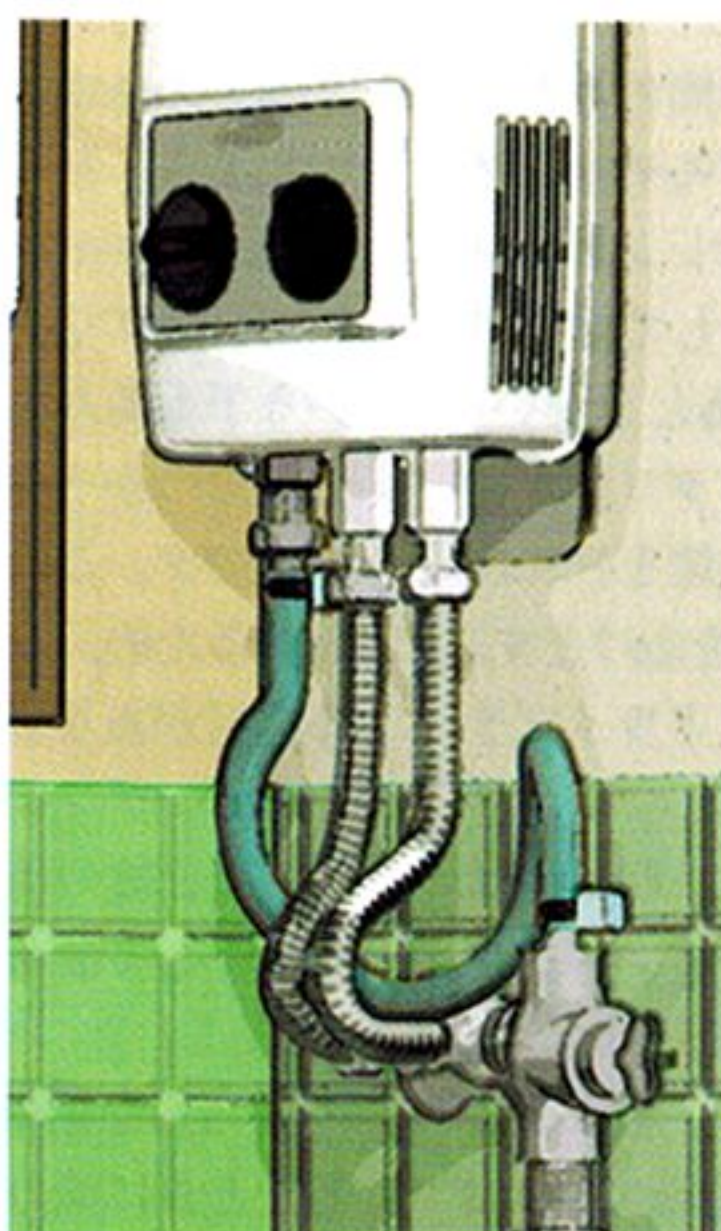



図56　ディテールアップ前の湯沸かし器

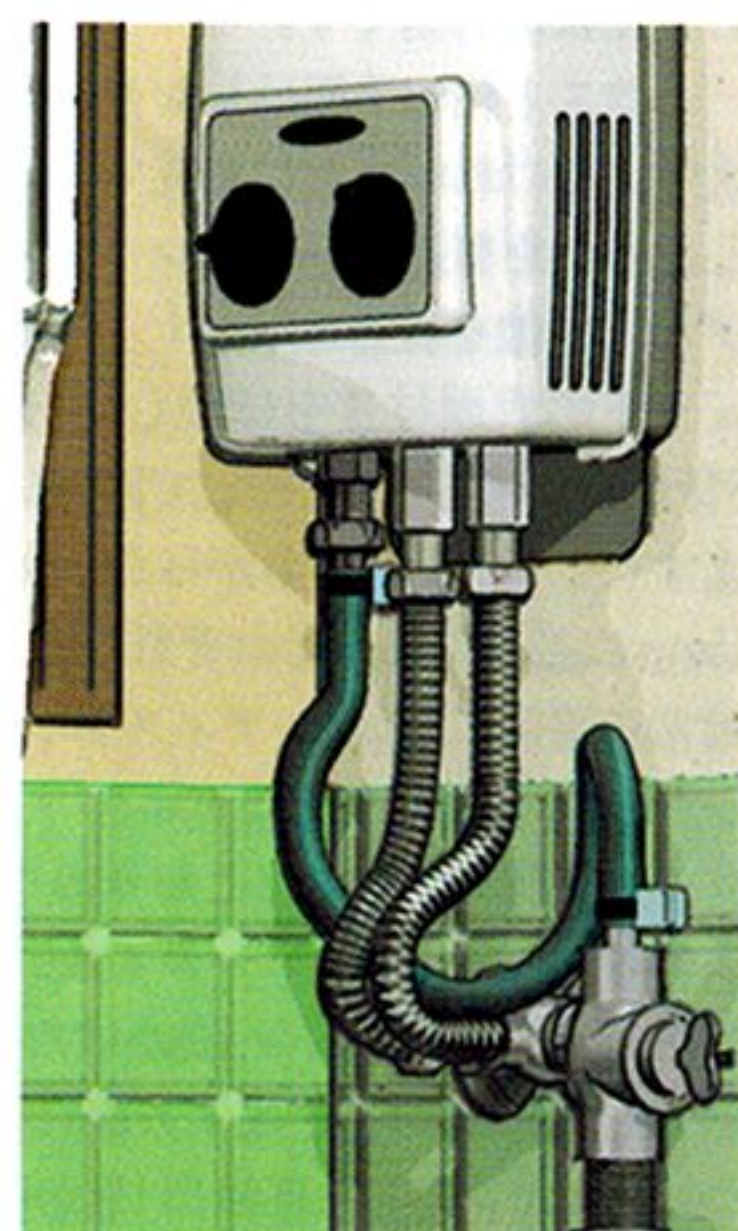



図57　ディテールアップ後の湯沸かし器

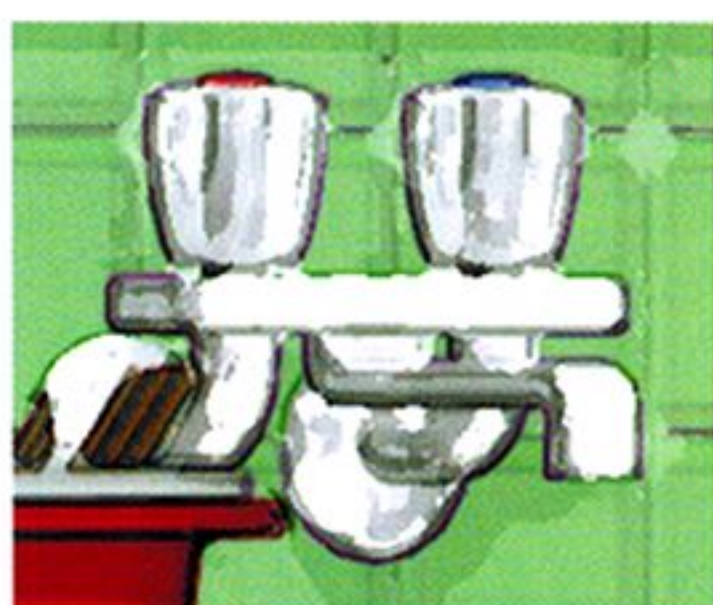



図58 ディテールアップ前の蛇口

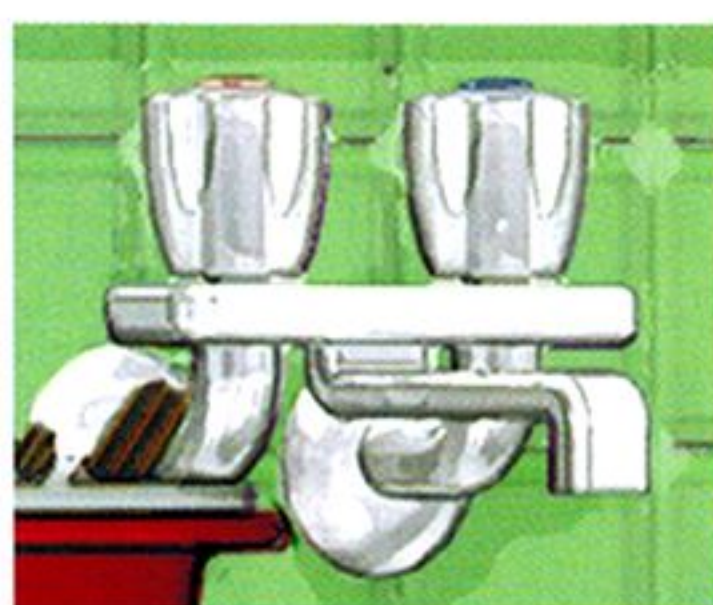



図59 ディテールアップ後の蛇口

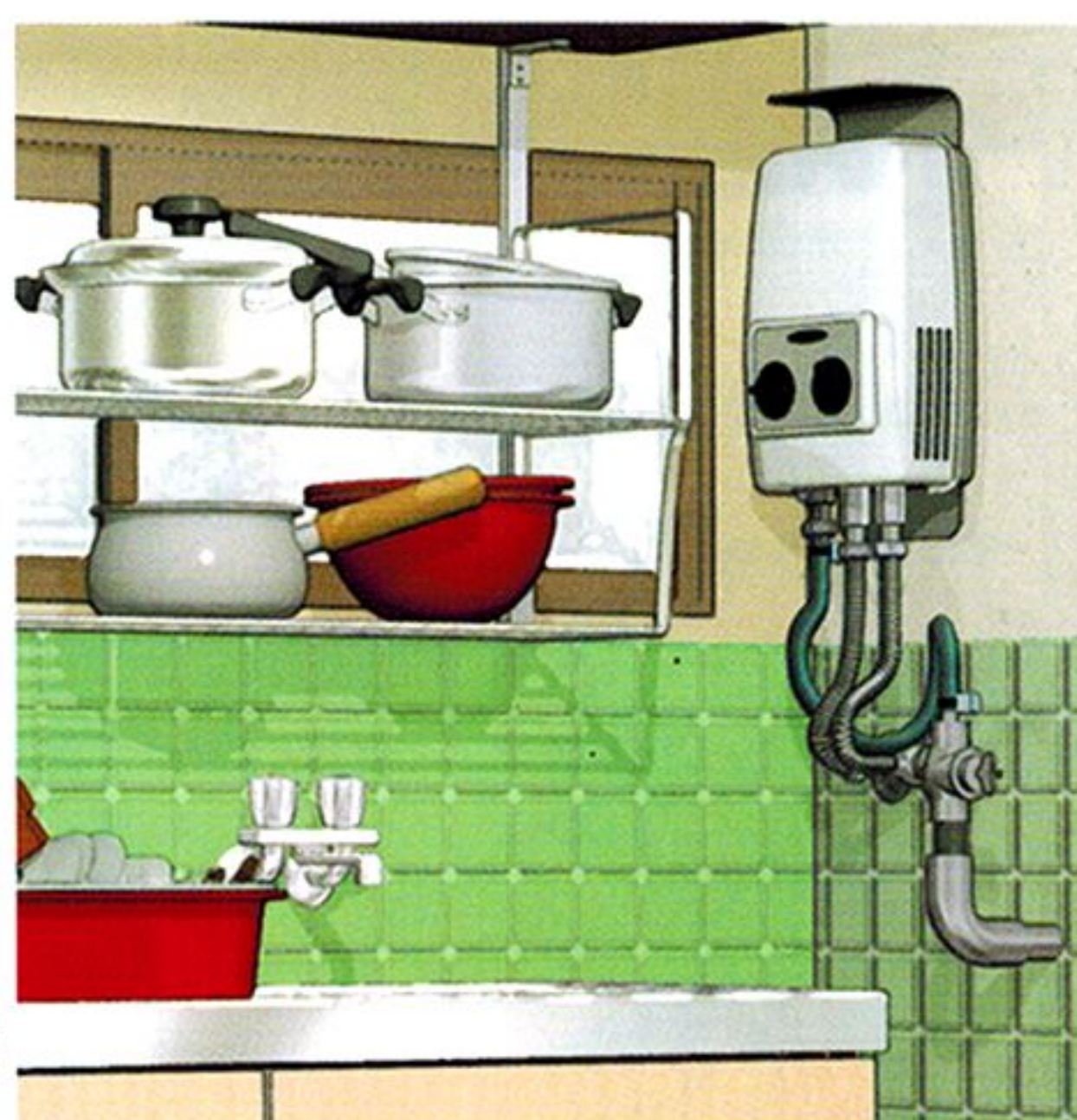



図60 レンダリングし直した画像を組み込んで背景のできあがりだ

図61 PhotoShop上のマジックワンドで範囲を選択し……

図62 細かいツールでマスク範囲を詰めていこう

図63 背景との大きさをあわせていく。この場合は背景を拡大した

図64 合成時に若干背景の彩度を落として人物に馴染むように調整

で描かれているため単調な感じがするのです。実際に絵を描くときのことを考えればわかりますが，細かいオブジェクトを描くときはそれに応じて細い線で描くはずです。そこで自分がこれを手で描くときにどこを細かく描くかを考え，Shadeに戻ってその部分だけをズームしてレンダリングし直すことにしました(図53)。ズームの際に気をつけなければいけないのは，画角が変わるとあとで背景画像の上から重ねられなくなるということです。画角を変えずに拡大するには，カメラウィンドウの「ズーム」を操作するときにOptionキーを押しながらコントローラ上を左右にドラッグします(図54)。オブジェクトを中央に収めるには「注視点」をチェックしてドラッグします。

このようにレンダリングし直して，ディテールアップしたものが図55aです。マスク付きですから合成も簡単です(図55b)。同様に水道蛇口などもディテールアップしました。図56・図57がディテールアップ前，図58・図59がディテールアップ後の画像です。やり方としては「塗り」の画像は図44の元画像を拡大して利用し，主線用画像だけをレンダリングし直すようにしたほうが，無駄なレンダリング時間が省けると思います。合成が終わったら背景は完成です(図60)。

人物と背景の合成

最終的にPhotoshop上で合成します。まず人物のほうにバックの白から人物を切り出すマスクを作成します。マジックワンドで背景の白を選択して(図61)アルファチャンネルに保存し，ペンツールで細かい部分を修正していきます(図62)。そうして作成したアルファチャンネルを選択範囲として読み込んでレイヤーマスクを作成します。その前に人物の絵をレイヤー化しておくことを忘れてはいけません。

完成した背景をレイヤー統合してドラッグ&ドロップで人物の絵に放り込みます。圧倒的に背景のほうが小さいので「編集：変形：拡大縮小」で背景を拡大します(図63)。もちろん人物のほうを縮小しても構いません。ここで人物の映り込みや影を背景に描き加えるべきですが，この絵の場合はこれ以上しつこく描き込む理由もないようなのでこのままにしておきます。あとは人物と背景

図65　こちらがそのまま合成してみたところ

図66　これが背景の彩度を調整しつつ合成した画像

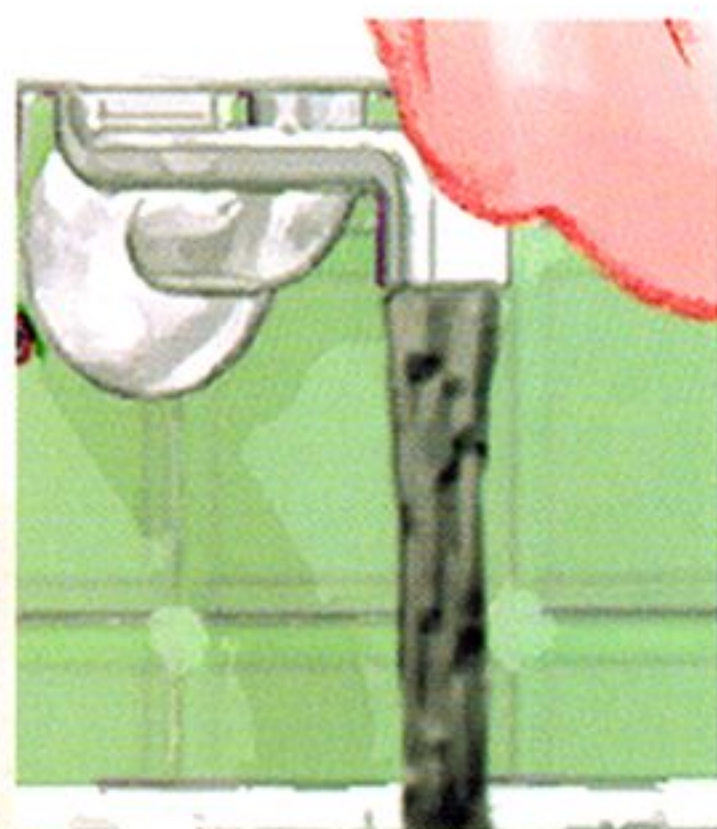



図67　仕上げに蛇口からの水を描き加えていく。ひとまず別レイヤーに描き……

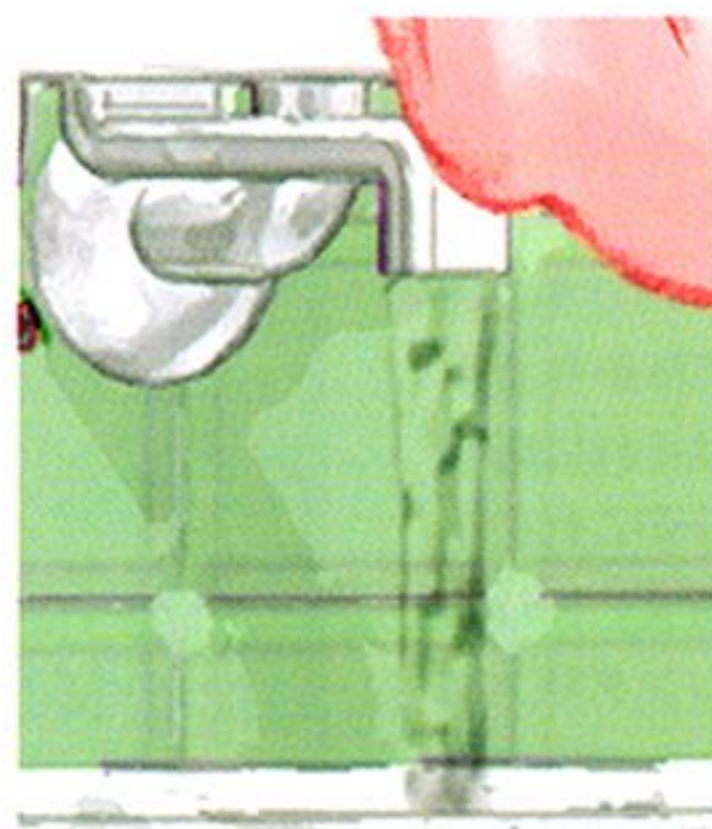



図68　透明度を指定して合成する

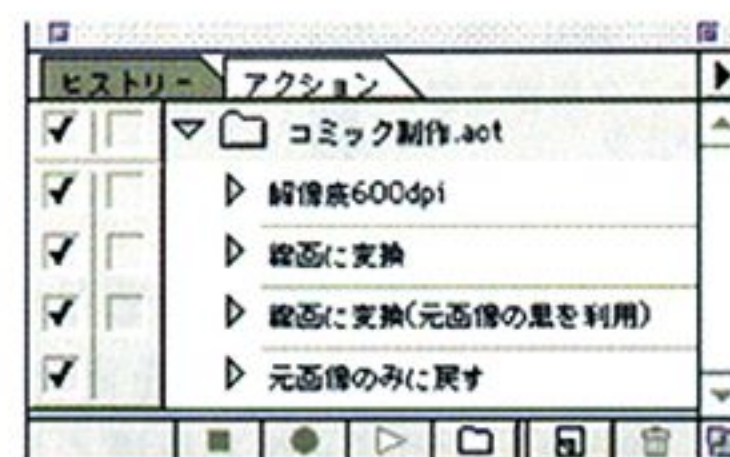



図70　PhotoShopの機能を自動実行するアクション機能

図71　こちらはShadeのQuicky

の描き込みのウエイトが同等ぐらいなので，人物が埋もれてしまわないように，背景レイヤーの上に「調整レイヤー：トーンカーブ」を加えて（図64a，図64b）若干背景の彩度を落としてみました（図65→図66）。

蛇口から出る水は別レイヤーに直接描き込んで（図67），「ハードライト・透明度50%」で合成しています（図68）。手に持っている皿も背景に合わせて3Dで作り直そうかとも思いましたが，3Dで作った背景と手描きの皿にそれほど違和感がないことを示すデモンストレーションとしてそのままにしておきます。

最後に人物の淡いタッチを強調するため（実際に淡いのは服の部分だけですが）エアブラシを使って白でぼかしてみました（図69）。余分な空白を切り落として完成です。

最後に

特に説明はしませんでしたが，背景のイラスト化を効率的に進めるにあたって助けになるものが2つあります。Photoshopのアクション機能とCEsoftwareのQuickeysという，いずれもアプリケーションの定型作業を自動化してくれるツールです。これによってPhotoshopでは主線抽出用の画像をボタンひとつで線画に変換してくれますし（図70Photoshopのアクション機能。Photoshopでの作業を自動化する），Quickeysではいまのところ，Shadeで画像サイズを何種類か変更する程度ですが（図71ツールバーが使えるのであたかもアプリに最初から備わってるツールのような感覚），アプリケーションを選ばずコントロールしてくれるので（実際には使えないアプリもあります）いずれはレンダリングから画像変換までの工程を自動化しようと思っています。

複数のアプリケーションにまたがってますます複雑化するグラフィックの作業では，こういった自動化ツールを使いこなすこともこれから必要になってくるのではないでしょうか。

図69　最後に絵の周りの部分をぼかして淡い感じに仕上げてみた

田中順子

Tanaka Yoriko

My Friend

制作環境
本体：自作PC/AT互換機
OS：Windows NT4.0
CPU：Pentium Pro/200MHz
搭載メモリ：128Mバイト
使用ソフト：LightWave3D 5.0

はじめに

Oh!X復刊2冊目おめでとうございます。私とOh!Xとの初めての出合いは，いまから5年ほど前になります。思い起こすと当時デザインの専門学校に通う学生だった私は，X68000でDoGAのCADやMatierを使ってCG制作に挑戦しようとしているところでした。このときの私は，パソコンというものがまったくわからず，起動から，ソフトのインストールまでを他人にやってもらわなければ，なにもできない状態でした。

当時はハード，ソフト，そして私の技量が乏しかったため，CG制作の苦労は大変なものでした。その頃もOh!Xの記事を読んでCGの勉強をしていましたが，いまとなってはいい思い出です。

今回は，そのOh!Xに記事を載せていただけるということで，大変喜んでいます。当時とは，環境も時代も変わってしまいましたが，私のCG制

図1　顔のラフスケッチを行います。デッサンの狂いや，線のはみ出しなどは気にせず，どんどん描き込んでいきます。本来この部分は自分のイメージを高めるためのところなので，少々雑なところは気にせずにデザインを進めていきます。デザインが決まれば，色鉛筆で色を塗ります

図2　体のラフスケッチを行います。体は，服装やアクセサリも含めてデザインを行うので，顔よりも時間がかかってしまいます。ここも少々の雑さは気にせずに描き込んでいきます。体も個性を出すためには重要なところなので手は抜けません。服装は作者のセンスを問われるところなので，少し緊張します。色きめも含めてデザインが決まれば，色鉛筆で色を塗ります

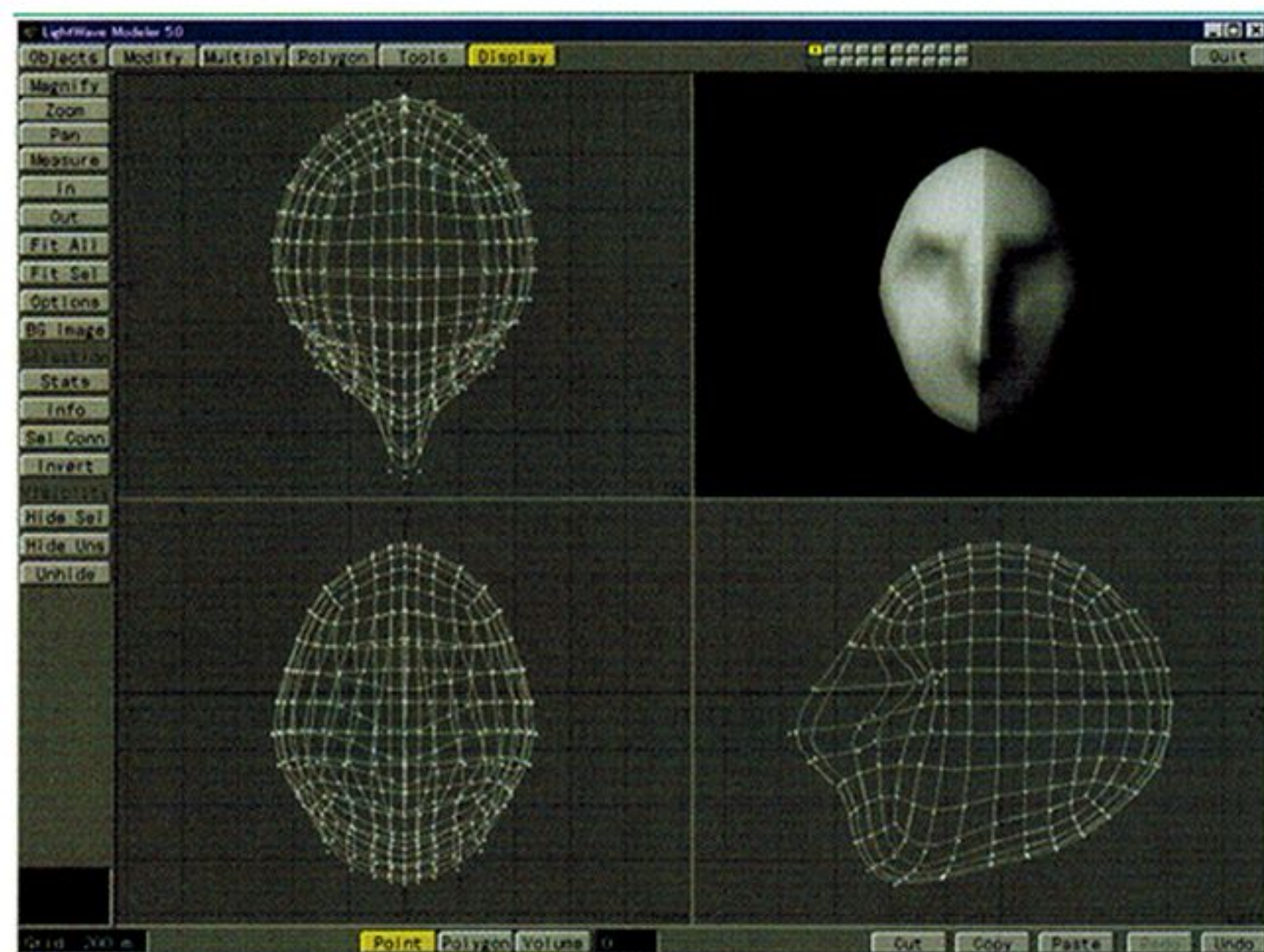
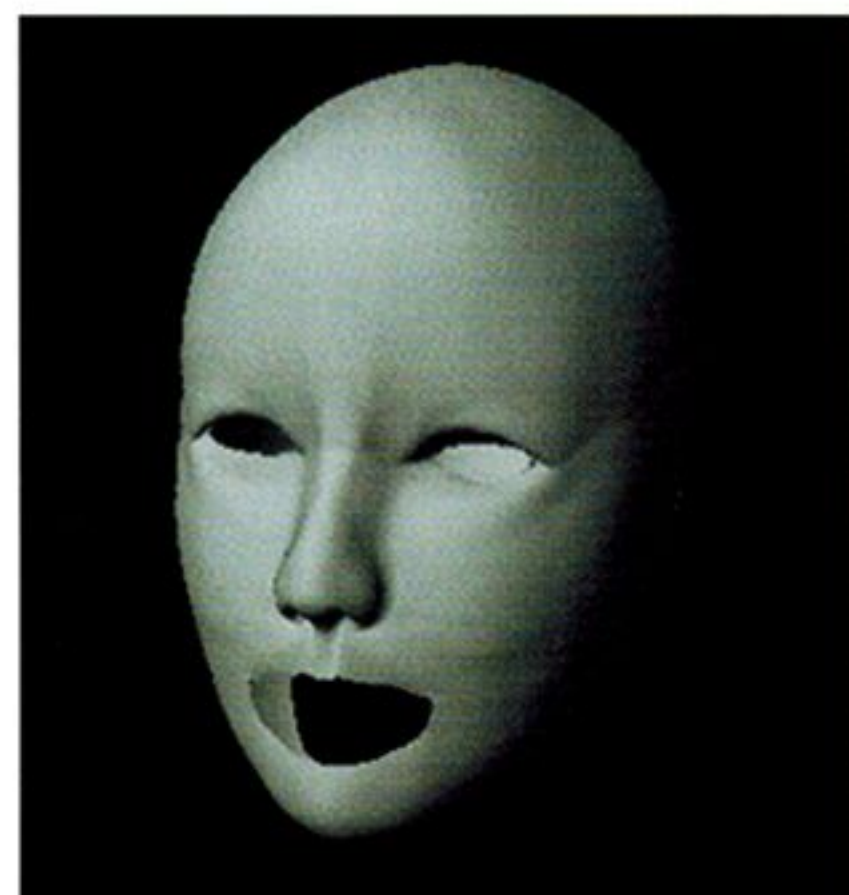
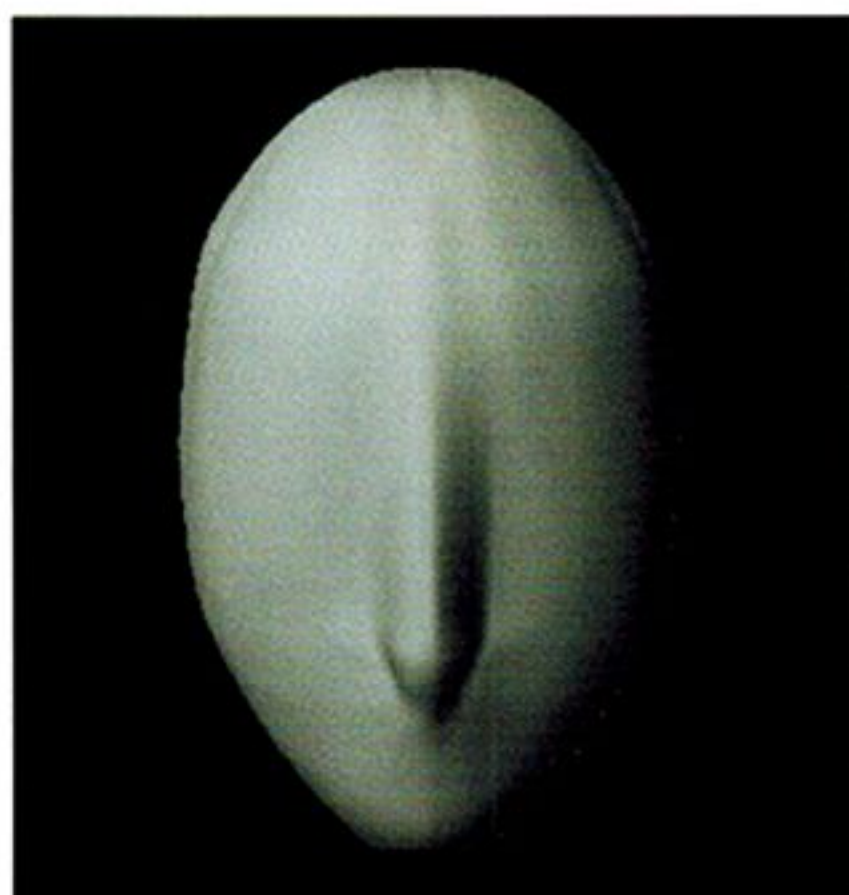



図3　キャラクターの顔ができるまでの制作過程です。まずは，大まかに輪郭と鼻を作ります。鼻を顔の基準にしているので，それを頼りに目と口の位置を決めて穴をあけます。微妙なラインを表現するために，最初の段階から少しポリゴンの数を増やしておきます。私の場合，ここでは特に技巧的なことはなにもやっていなくて，LWの各ツールを使いながら，ただひたすら細部にいたるまで作り込んでいきます。根性だけの世界です。男性キャラクターも同じように制作していきます。ところで，LW3Dでは，このように人物の微妙なラインを作りたいと思ったとき，ちょっと困った点が出てきます。LW3DのNURBSモデラーは，曲面は安定する半面，ラインが少し硬くなるようです。そのため，自分の思ったような微妙なラインを作ることがなかなかできません。そのため，作り込むにつれ，通常より制御点を増やしていき，微妙なラインの表現を行おうとしています。こうすると曲面の安定性はなくなってきますが，微妙なラインの表現が行えるようになります（ただ，それでもまだ満足のいく曲面を描くのは私にとっては難しいようです）

作に関しての紹介をさせていただきます。

作品のコンセプト

ここ最近，女性キャラクターばかり作っていたので，久しぶりに男性キャラクターを作りたいと考えているところでした。そこで今回は，ちょっと自信はなかったのですが，女性キャラクターと一緒に男性キャラクターも参加させてみました。仲のいい男の子と女の子という雰囲気を出したいということが今回のテーマです。

ところで，私の創るキャラクターの傾向を見ると，だいたい20歳から24，5歳くらいが好みの年齢みたいです。ですから，今回の作品も，少し大人っぽく，仕上げたいと思っています。その感じが出ていれば，とても嬉しいのですが，いかがでしょうか。

デザインを考える

これはCGをする人だけに限らず，もの作りに係わる人はみんな一様に苦しむところだと思います。しかし，もちろん，これからどんなものができるのか，とてもわくわくして楽しみなところでもあります。この創造的な部分がクリエータを自負する(？)私としてはいちばん重要であると思えます。

私の場合もここの作業では，時間をかけて，デザインをします。ここであわててしまうと，後でどんなにがんばっても作品そのものが中途半端な状態で終わってしまいます。

この部分でキャラクターのデザインや全体の構図を考えるときは，スケッチブックにラフを描きます。ラフを描くときは，思考を止めたくないので，見た目の形にこだわらず，どんどん描き込んでいきます。デッサンが歪んでいようが，気にせずにデザインを進めていきます。最終目的は3DCGでの作品なので，それがきちんと仕上がりさえすればいいのです。

さて，次に具体的なキャラクターデザインの話に入ります。女性キャラクターのほうは，少し大人っぽさを強調するために，服の色は，トーンを

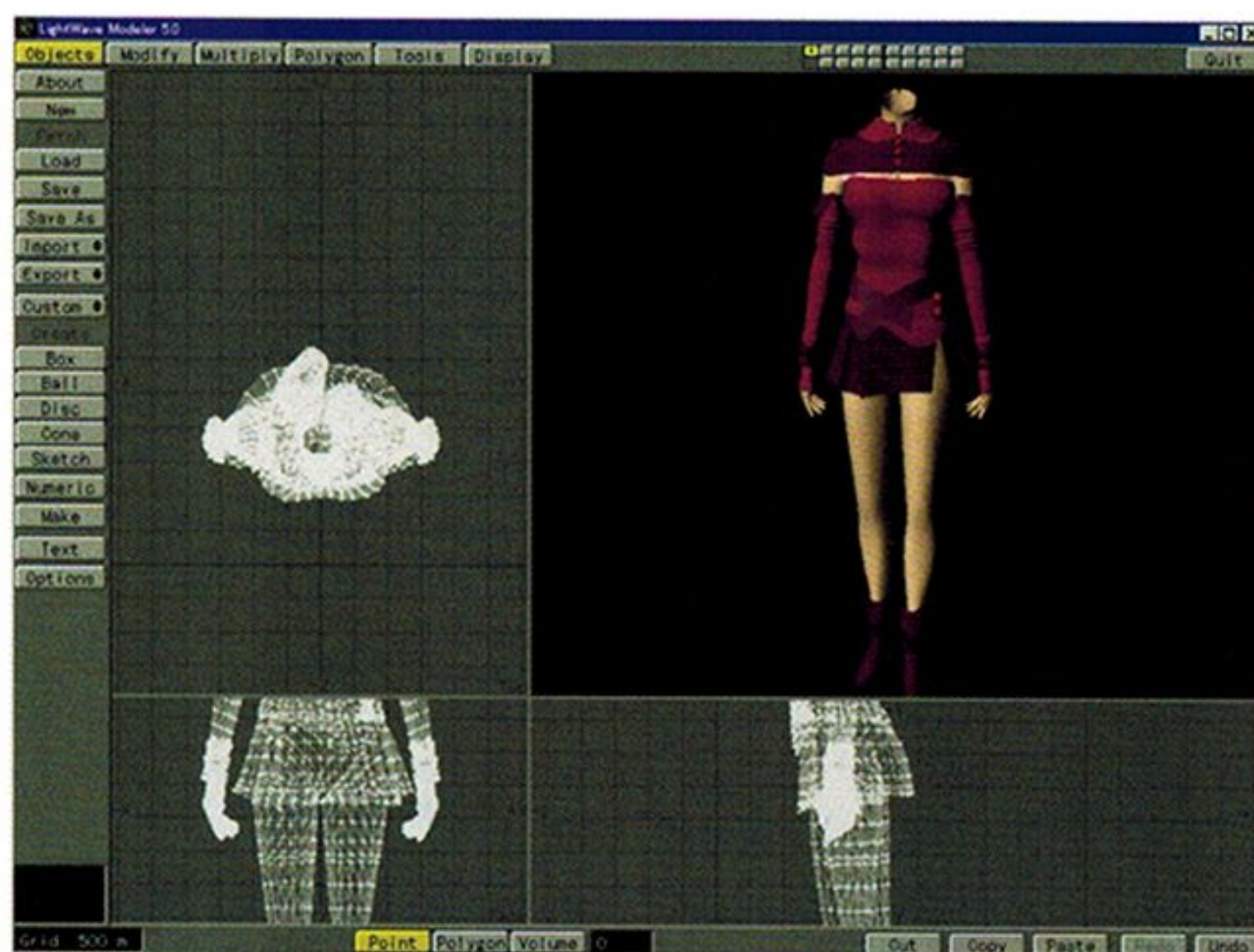
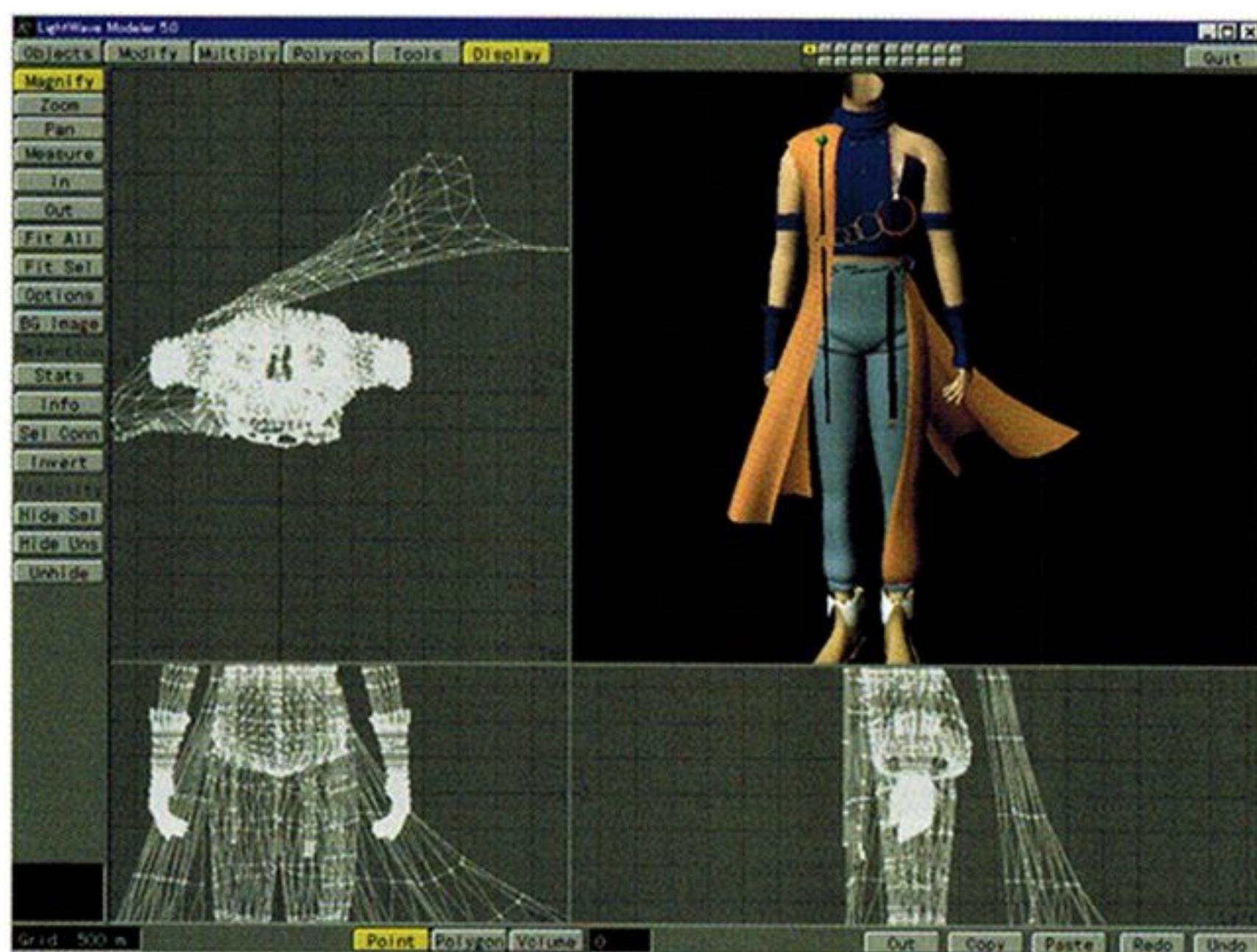



図4　ラフスケッチを基に作った基本のポーズです。これでとりあえずキャラクターが完成しました。あとはここから，キャラクターに表情やポーズをつけ，最終的な構図の制作に入ります

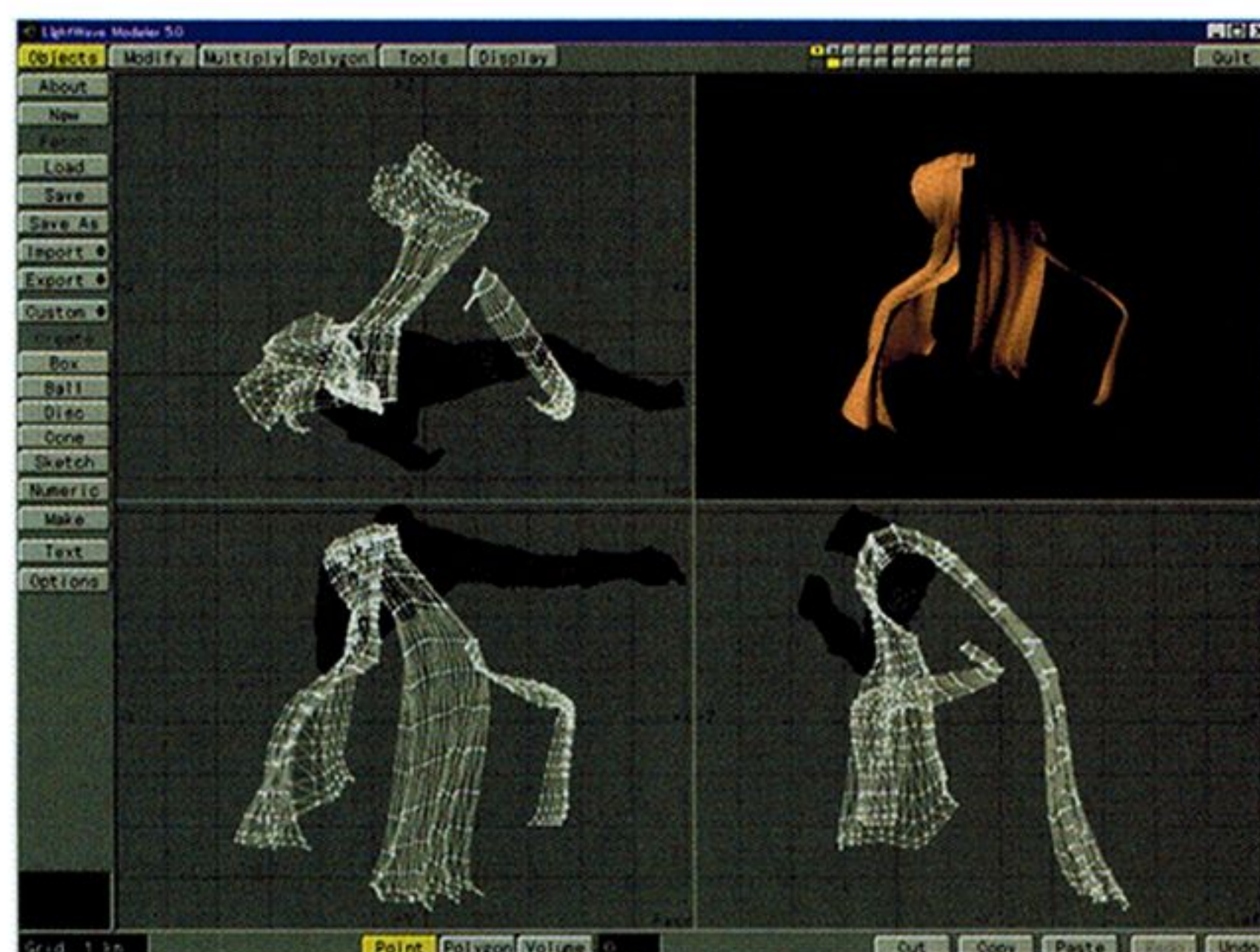
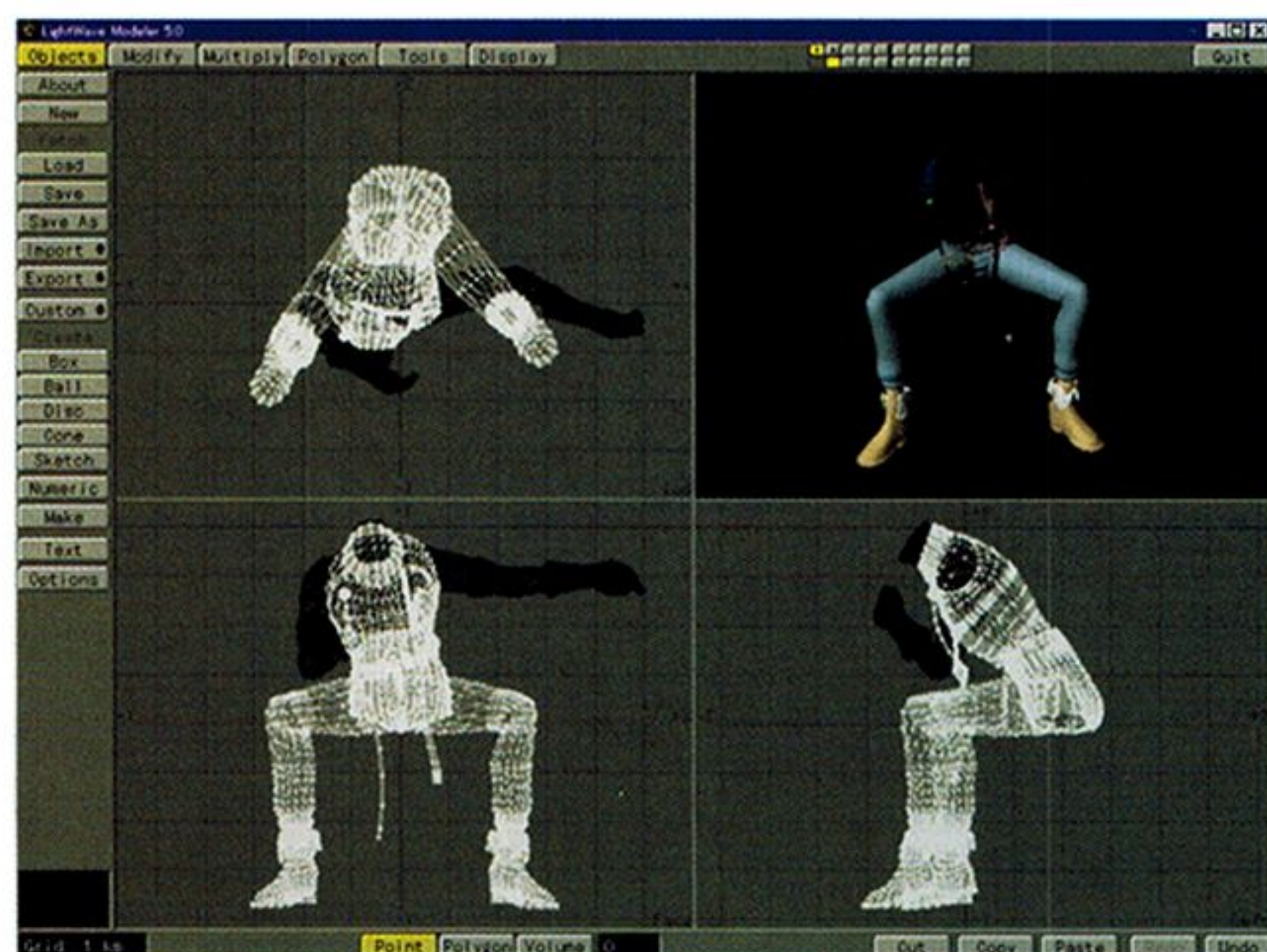
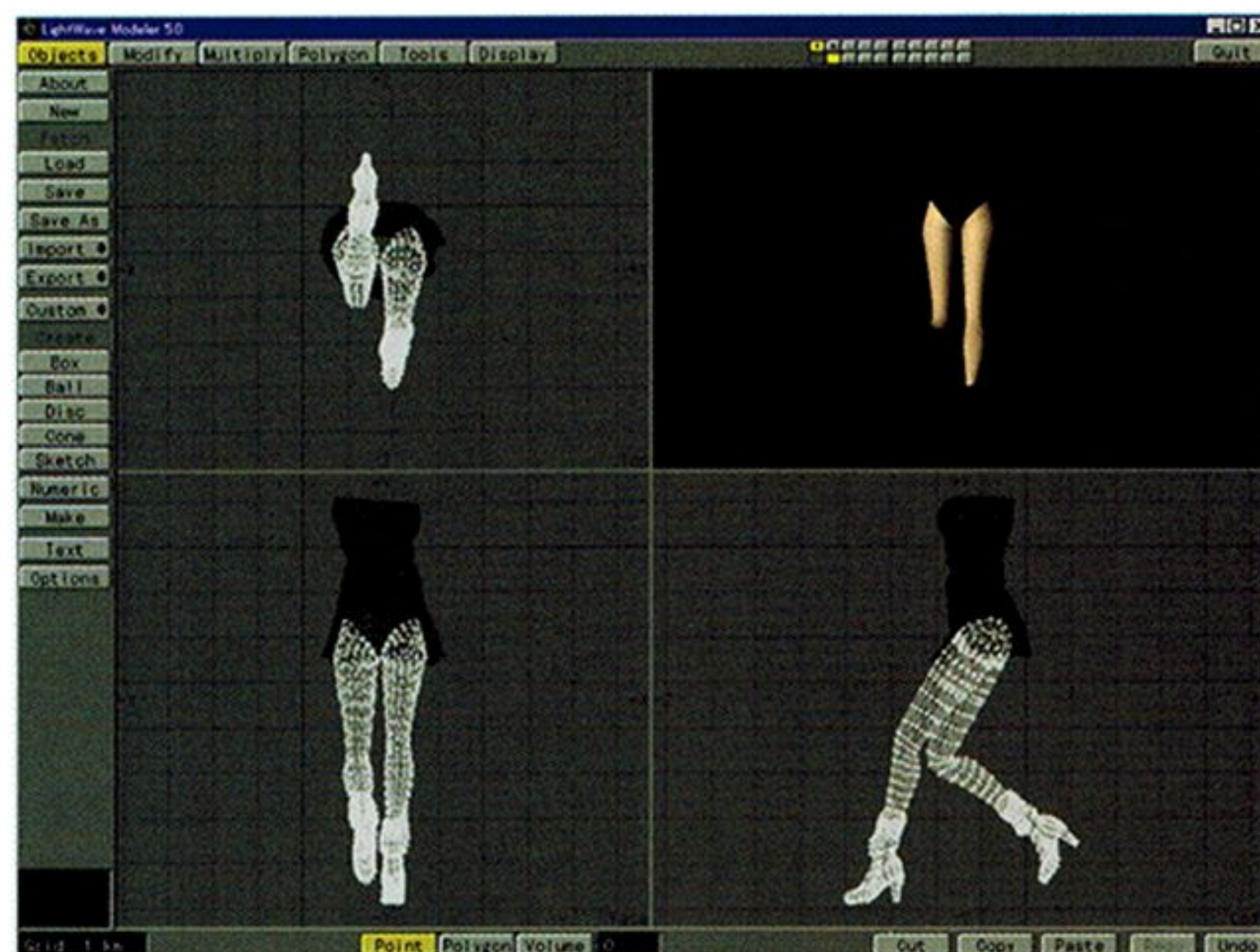
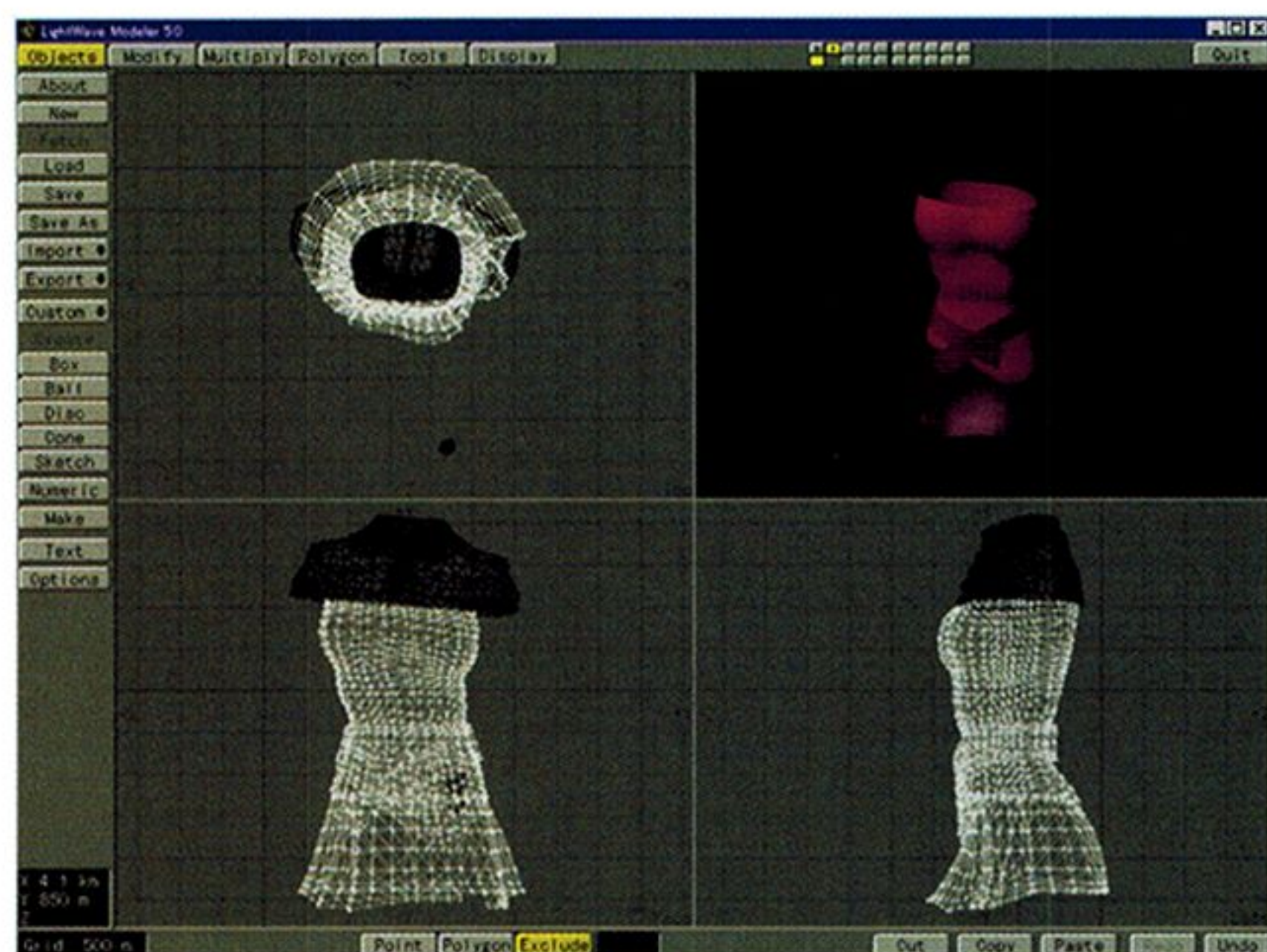
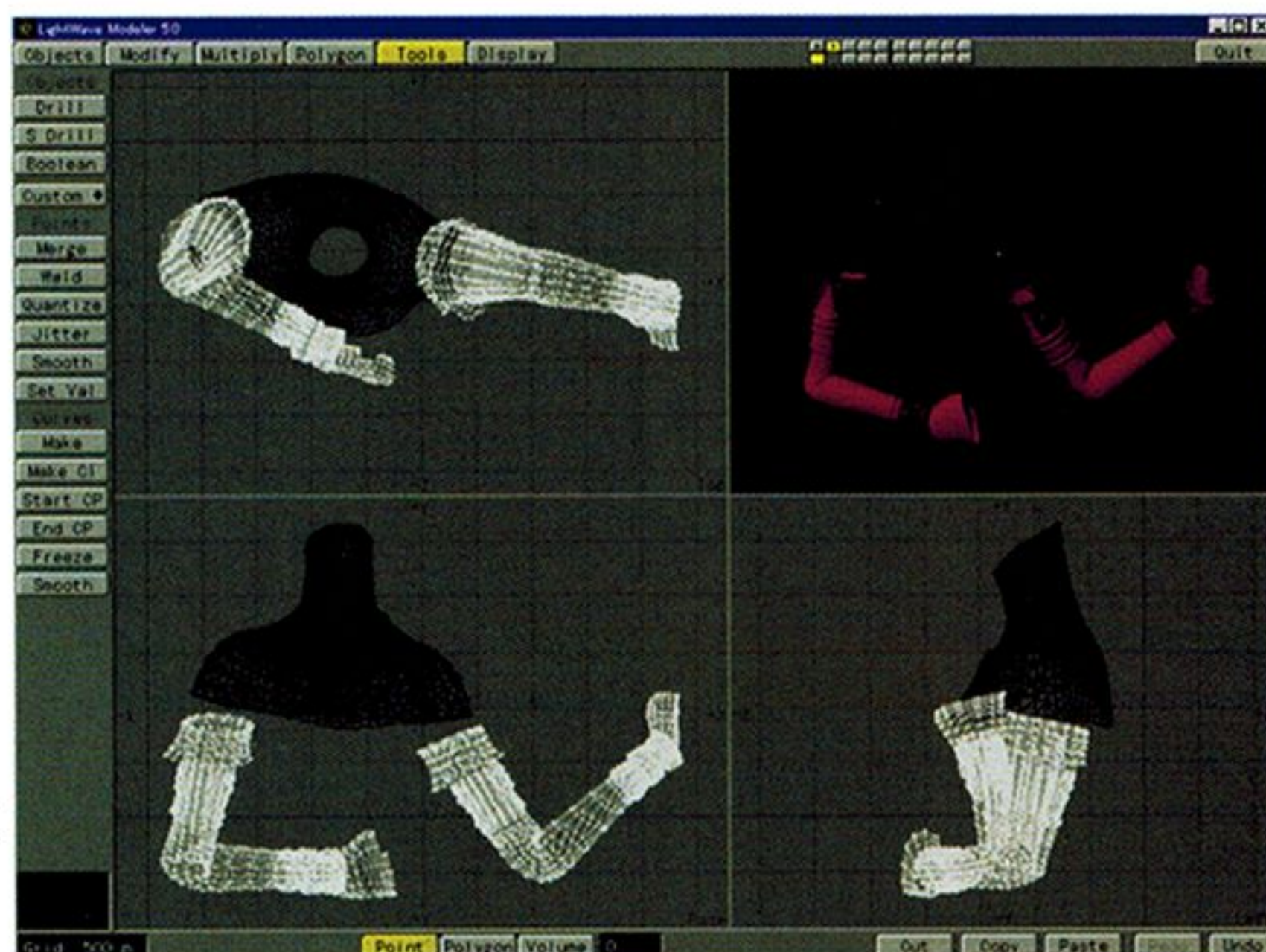
落としたワインレッドにしました。そして，少しカジュアルな感じも出したかったので，胸元を少し見せ，スカートをミニにしてみました。プロポーションに関しては，足の長い女性をイメージしていたので，雑誌に載っていたモデルの体型を参考にしました。

男性キャラクターのほうは，少年のような心を持った男性に仕上げたかったので，少し，やんちゃな雰囲気にしています。そして，おどけた表情で彼女にちょっかいをだしているようなポーズをとらせることにしました。服装は，ジーンズとタンクトップをベースにして，独自にアレンジしています。背景は，人物を引き立たせるために，あまり小物を置くのをやめ，シンプルに仕上げました。

キャラクターを作る

人間の体でいちばん重要な部分はどこでしょう。初めて人物の絵を描こうとするとき，ほとんどの人は，まずは顔から描きます。手や足から描き始める人はいませんよね。それは，顔が，その人がその人であることを示すいちばん大切な部分だからです。その表情は多くのものを伝えることのできる不思議な魔法のようなもので，人間の素晴らしさを表現するためには欠かせないものです。そのため表情をうまく表現することは大変難しいことなのです。

さらに，人物には顔だけではなく，体もあります。いくら顔をうまく描けてもそれにつりあうボディが描けなければ，人物の作品としては未完成になってしまいます。体にも当然いろいろな表情

図5　男女のキャラクターにポーズをつけます。各部品ごとに細かく作り込んでいきます。LWのレイヤー機能を使って，ほかの部品とのバランスを取りながら，作業を進めていきます

があり，人の魅力を表現するのに必要なところなのです。

次に具体的な制作の話をします。キャラクターを作るときは，まず全体の形を大まかに作ります。最初から細かいところを作ろうとするより，デッサンの歪みなどが見つけやすく，全体のイメージを壊すことなく修正できるからです。

私は，まず顔を作り，それにあったプロポーションを決めて，胴体を作っていきます。そして，顔と胴体をあわせて，今度は，修正と細部の作り込みをしていきます。ここでは，私は見せるための修正といっているのですが，見た目をよくするための修正をします。満足するまで続けます。人物の作り方に関しては，写真や，デッサンの本などを参考にしています。

背景を作る

背景は全体のイメージを決めるところなので構成がとても大変でした。実は背景の制作は私のもっとも苦手とする部分なのです。特に背景の配色には気を遣いました。色の組み合わせはとても難しく，どの色をどのバランスで使うかとても決めにくいところなのです。なぜなら，背景に使う色は，作品のイメージに大きな影響を与えるからです。私が背景を作る場合，背景になにを置くかではなく，背景にどの色を置いていこうかということに重点を置きます。

そのため，まず背景もラフスケッチを行いイメージを膨らませていきます。最初に頭の中でこの作品のイメージカラーを決めます。そして，そのイメージした色を塗り，それから，色のバランスと色の構図を考えながらほかの色を決めていきます。そして，キャラクターをどのように配置しするかを決めます。それにあわせて，周りのオブジェクトを決めて配置します。

今回は，落ち着きのある雰囲気の背景にしたかったので，シンプルに仕上げました。

仕上げ

モデリングができ，構図も決まれば，LightWaveのレイアウト上での作業が主になります。カメラを決め，ライティングを決めていきます。

ライティングは，ライトの当たり方ひとつで表情や，全体のイメージがガラッと変わってしまいます。ここは，コンピュータ上での作業だけでなく，実際に紙の上で設計をしたり，ものを配置したりしてイメージを作っていきます。

そして，それを基本に何度も慎重にレンダリングを繰り返します。ここで綺麗な影が入らないとモデリングの段階まで戻って修正しなければならない場合があります。これは，実際のシーンでの物体配置やライティングによる配光を行うまで，気がつかなかったモデリングの微妙な狂いがあるからです。

私は，あまりライティングの設定がうまくないため，かなり苦戦しながら決定しています。ときには，影が決まらないのはモデリングの狂いによるものか，それとも，ライティングのまずさによるものかわからない場合があります。そういうときは，試行錯誤を続けて，多くの時間を消費し修正を行うのです。

これが一流の写真家ともなると，作品にすごく綺麗な影や陰を入れることができます。私もそれを見習おうとしているのですが，なかなかうまくいきません。

モデリングやライティングが決まれば，あとは，解像度を上げてレンダリングを行い，細部を確認すれば完成です。

おわりに

今回は，こんな私の話におつきあいいただきましてありがとうございました。私自身3DCGのエキスパートというわけではないので，あまり参考にはならなかったかもしれません。そのわりにかなり偉そうなことを書いてしまって少し反省もしています。しかし，昔からファンだったOh!Xに私の書いた記事が掲載されるというのは，この上ない名誉に感じています。今後もOh!Xとそれに携わる皆様のご活躍を期待しています。

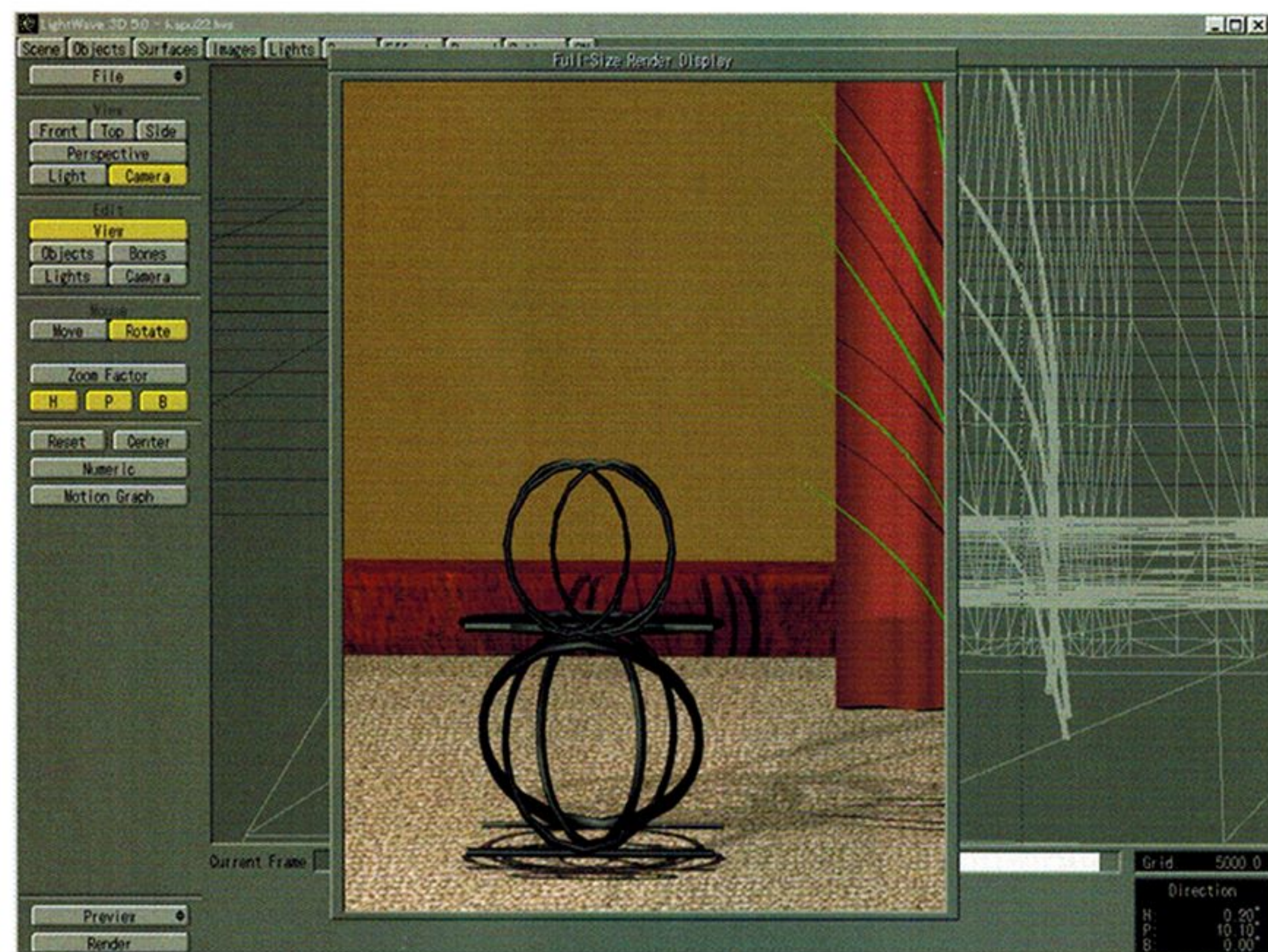



図6　背景は，人物を引き立たせるようにシンプルに仕上げています

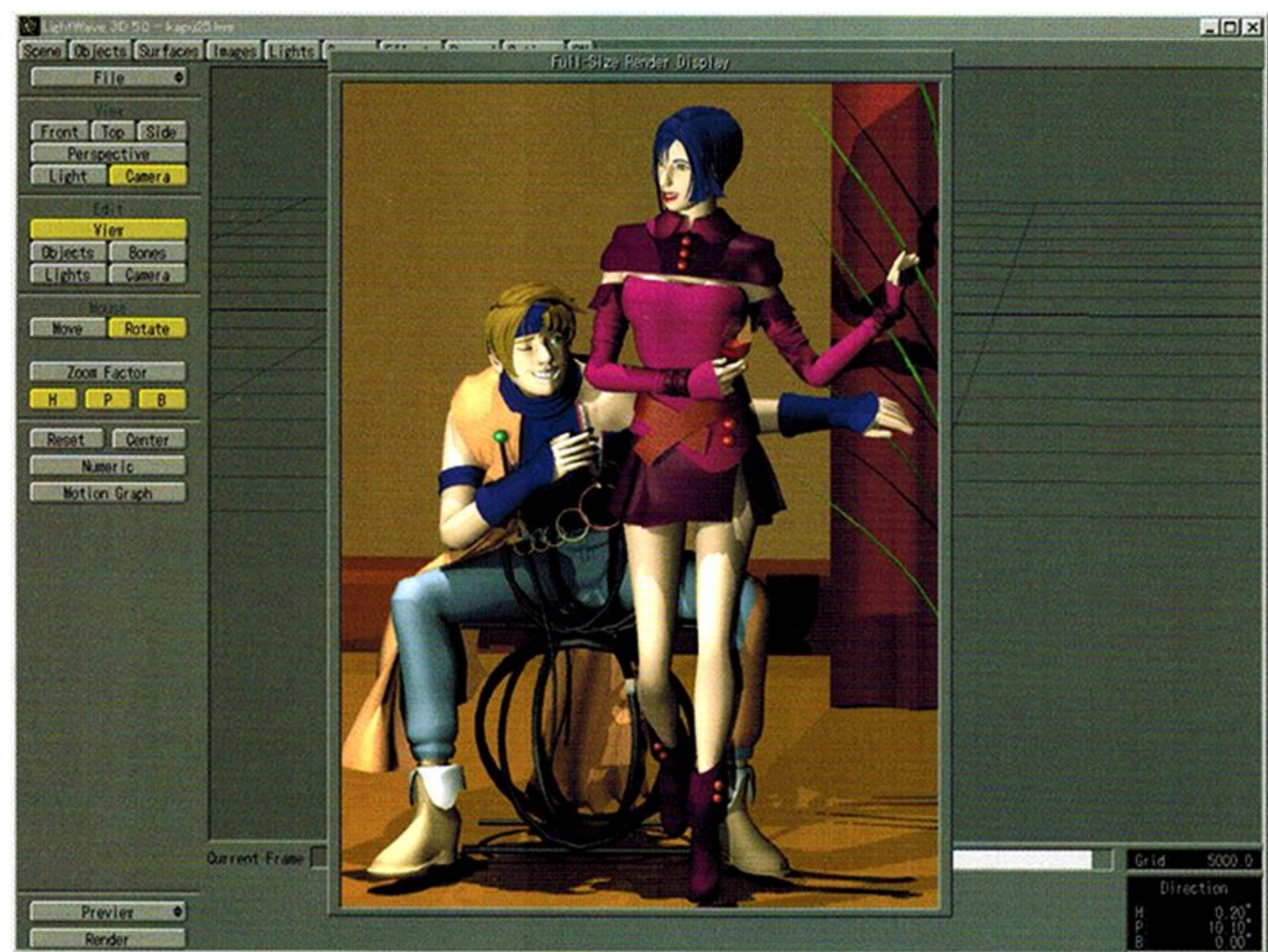



図7　マッピングなしの状態でのレンダリングです

由水桂

Yoshimizu Kei

Lightwave3Dによるキャラクターモデリング
映画のように

今回は眼に力のあるキャラクターで，
映画の1シーンのような，印象的なスタイルを狙ってみました。

制作環境
本体：PC/AT互換機 Pentium II/300MHz
メモリ：192Mバイト

使用ソフト
OS：Windows 95
LightWave3D 5.5

リアルな顔を制作する

最近は，3Dソフトのモデリング性能の向上も手伝ってか，CGでキャラクターを作るのが流行っている様子ですが，そんななかでも，「リアルな」キャラクターを作るのは，どんなキャラクターにも応用の利く技術ではないでしょうか。

ここでは，Lightwave3DのMetaNURBSを使った「リアル」なキャラクターのモデリングについて具体的にご紹介したいと思います。

Lightwave3Dでのモデリング

図1　これがLightwave3Dのモデリング画面です。今回はMetaNURBSという曲面を持ったもののモデリングに非常に有効な手法をご紹介します

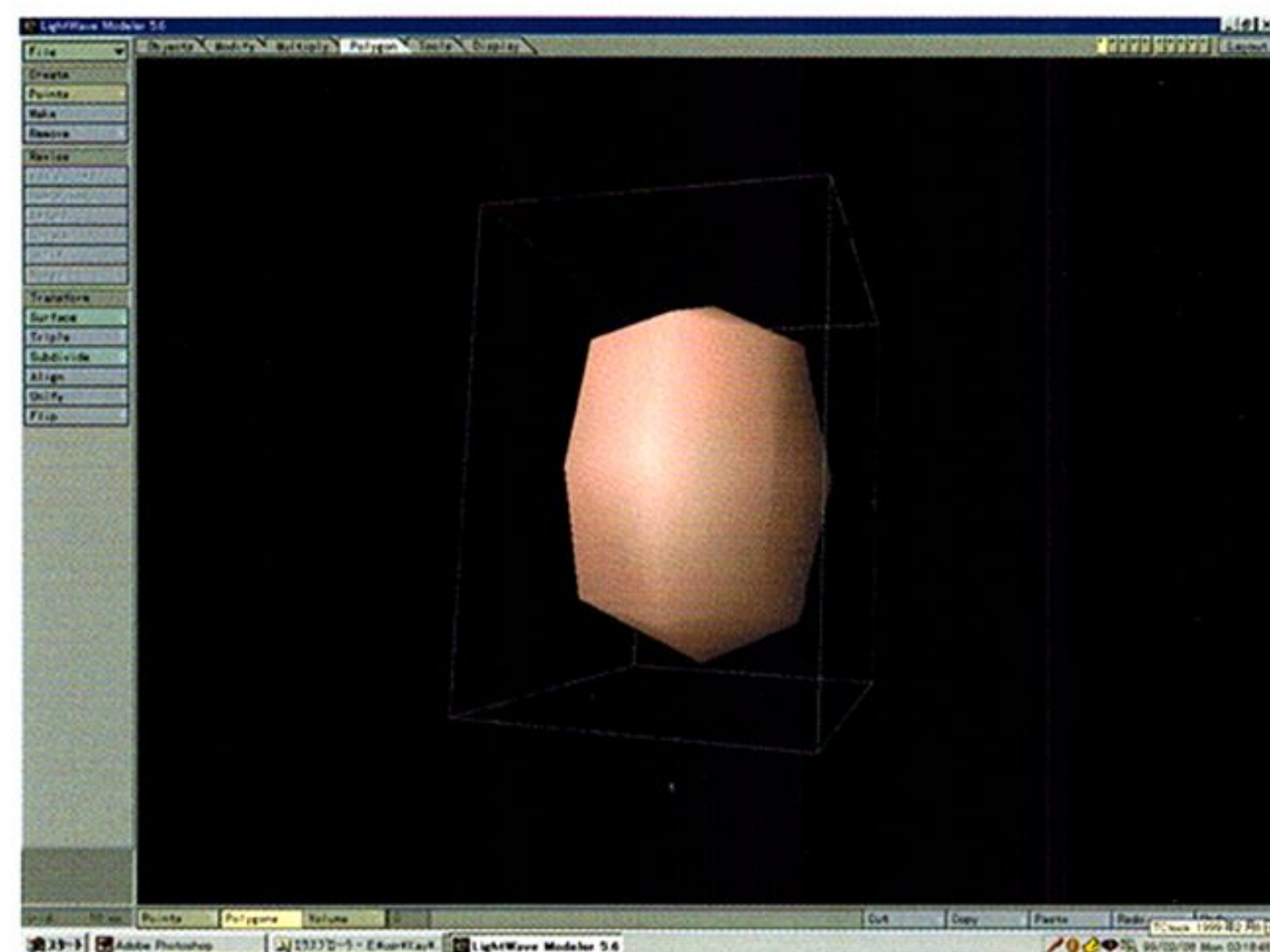
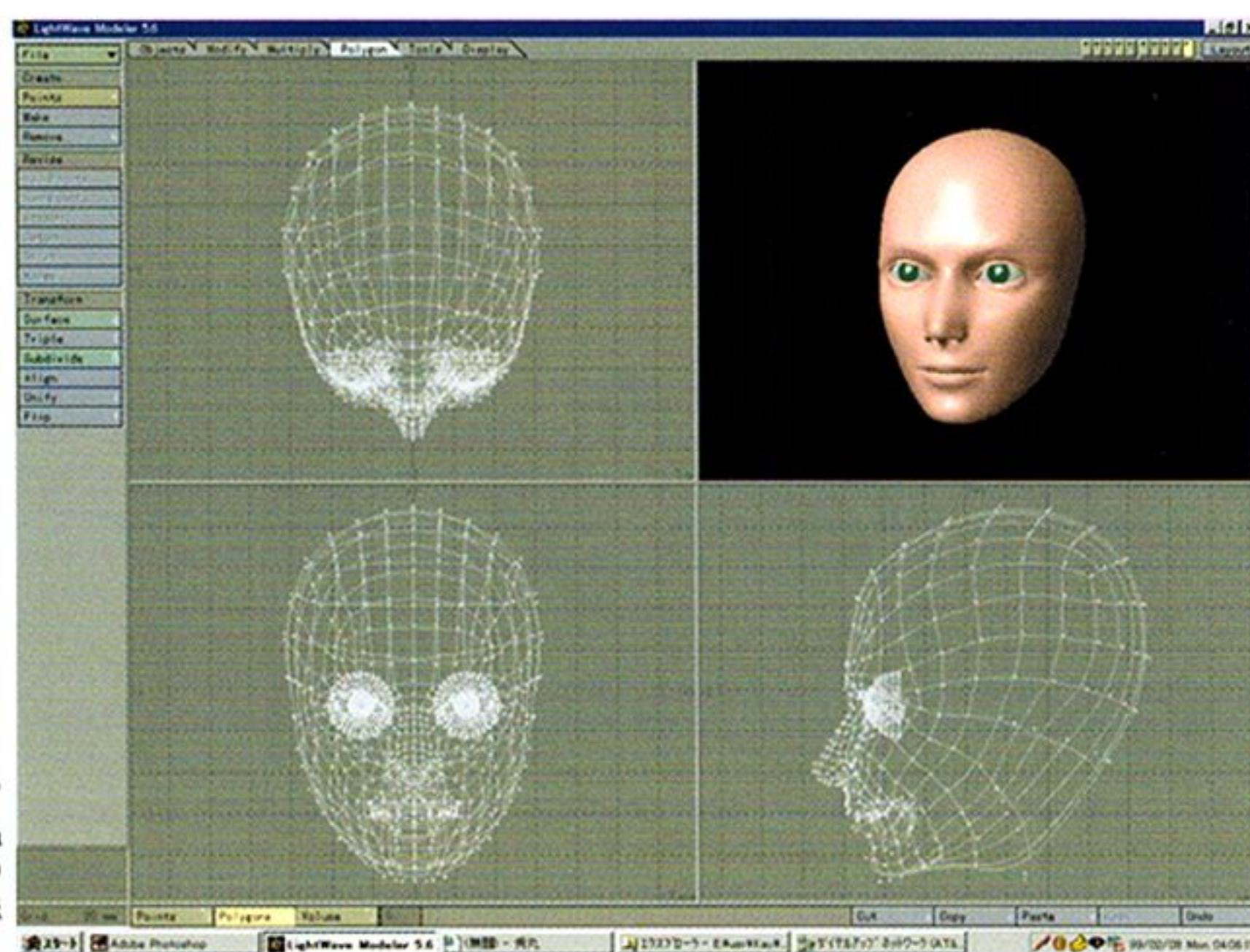



図2　BOXを作り，TABキーを押します。するとMetaNURBSオブジェクトが内側に現れます。外側の制御ポリゴンを操作していくことによって内側のMetaNURBSオブジェクトを粘土のように成形していくのです

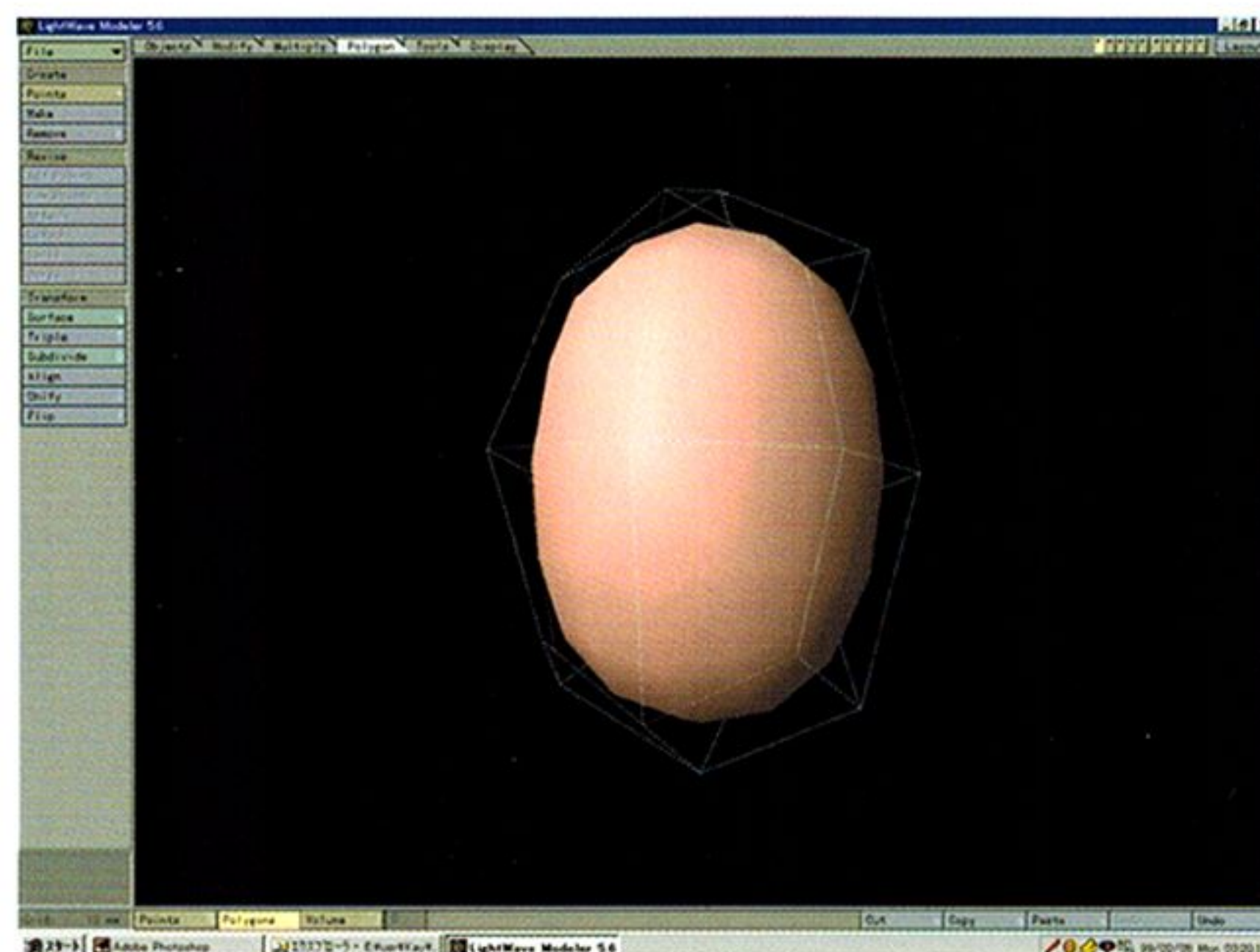



図3　Subdivideでラウンディングしたところ。制御ポリゴンがBOXより小さくなったにもかかわらず，MetaNURBSオブジェクトは大きくなっているのがわかります。制御ポリゴンが密集するとオブジェクトが制御ポリゴンに接近するのがMetaNURBSの特徴なのです

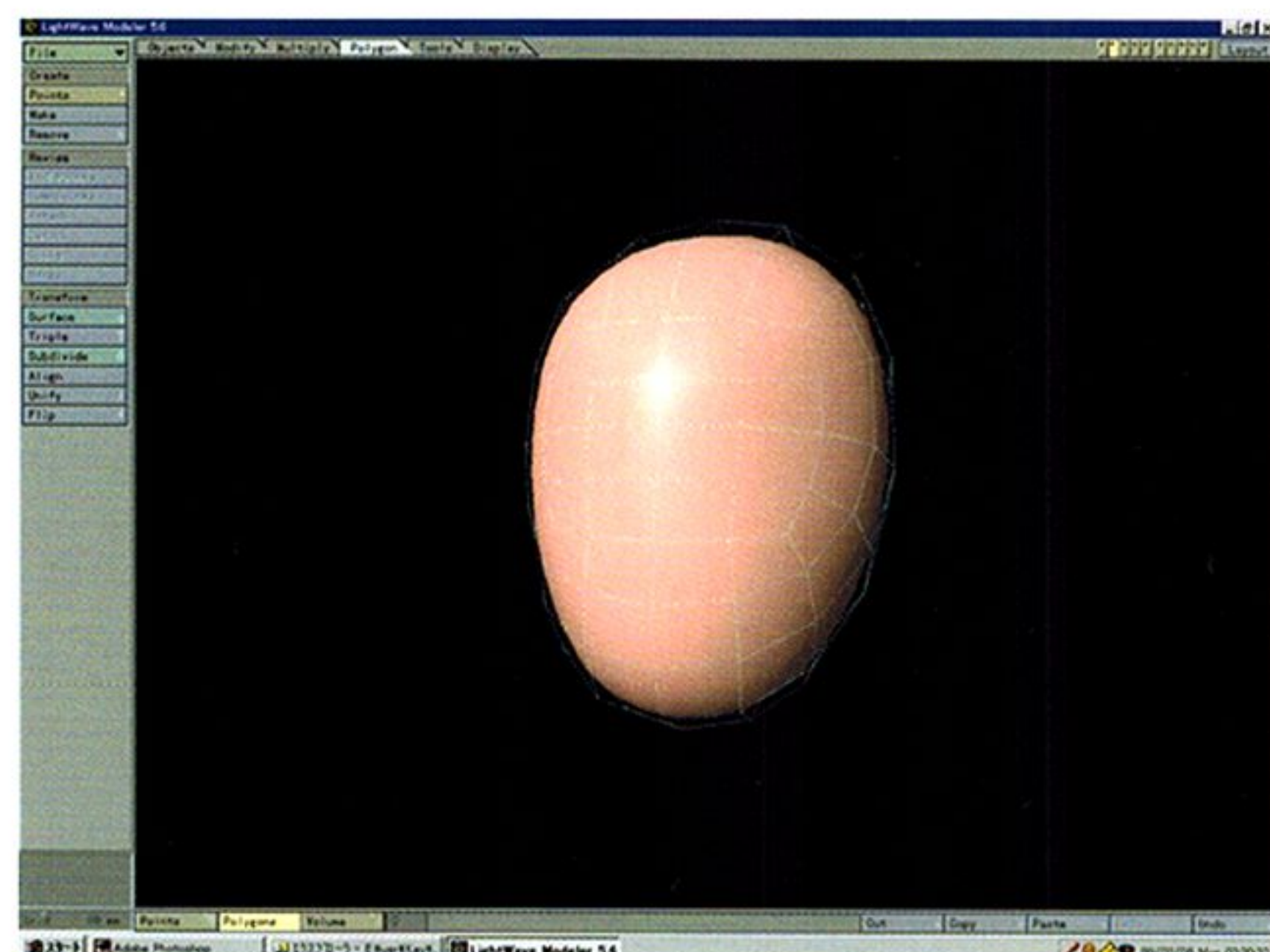



図4　さらにSubdivide。MetaNURBSの制御ポリゴンは，三角形もしくは四角形で構成されていなければならないので，ポリゴン数を増やすにはSubdivideやSmooth Shift，Bevelといった四角形を作り出すツールを中心に作業します。また，ここでは制御点の多く必要な顔の部分に後ろからポリゴンを寄せるようにして密集させています

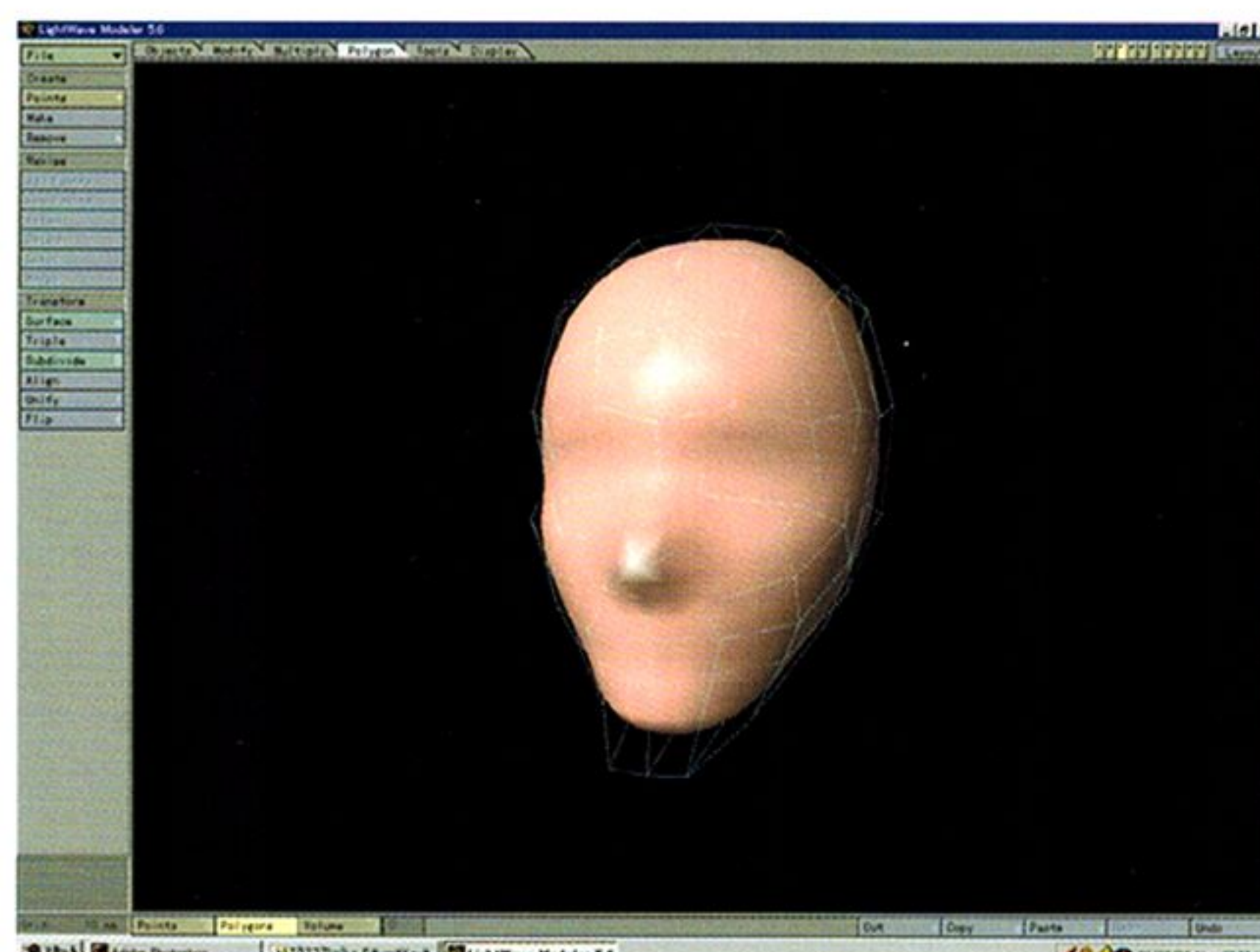



図5　いよいよMagnetなどのツールを使って形を作っていきます。制御点を押したり引っ込めたりしてあたかも粘土をこねているかのように顔の形を作っていきます

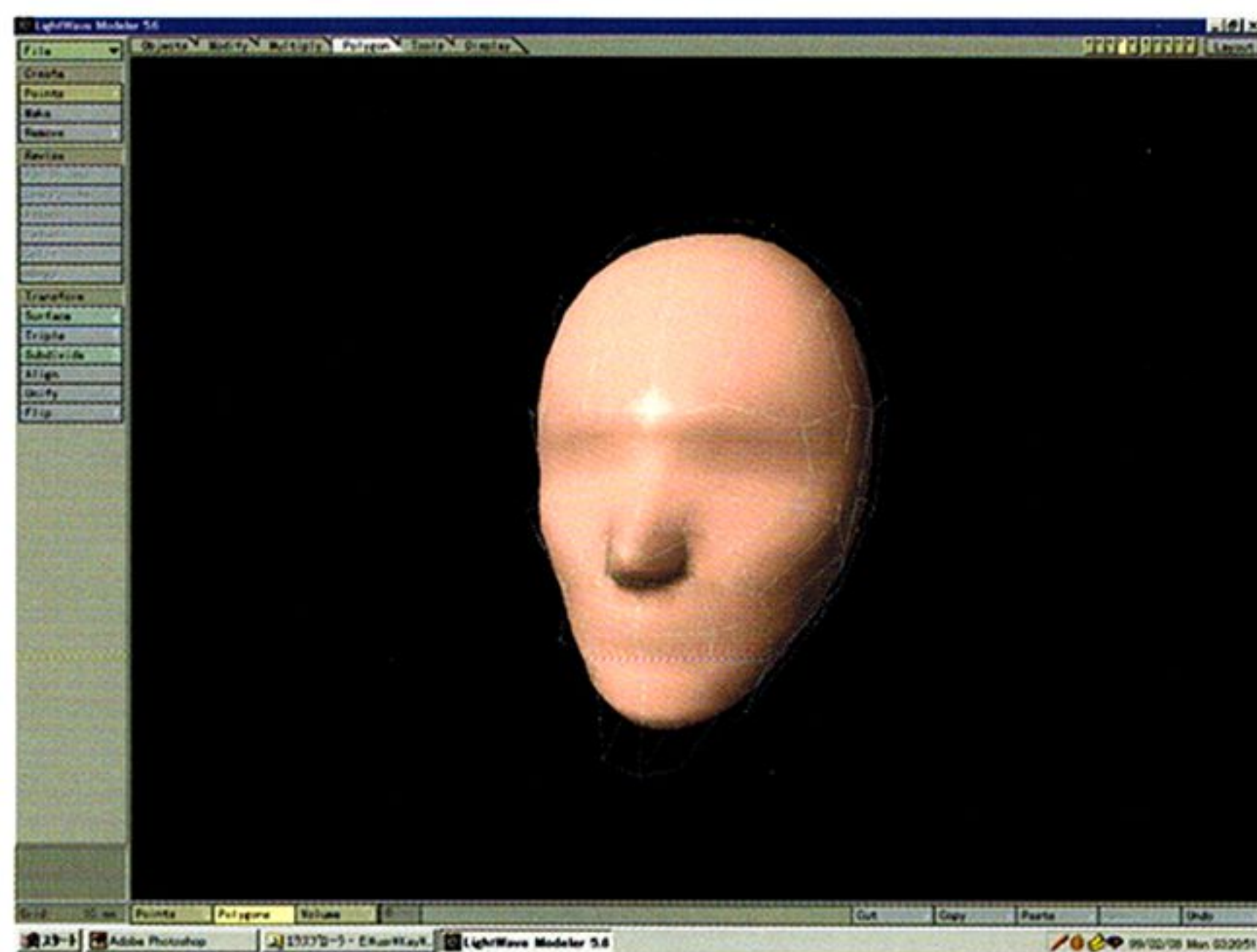



図6　ここで鼻のディテールを出すため，鼻の下の部分を押し出し，形を成形。とりあえず顔の半分だけモデリングし，Mirrorでもう片方も作ると便利です

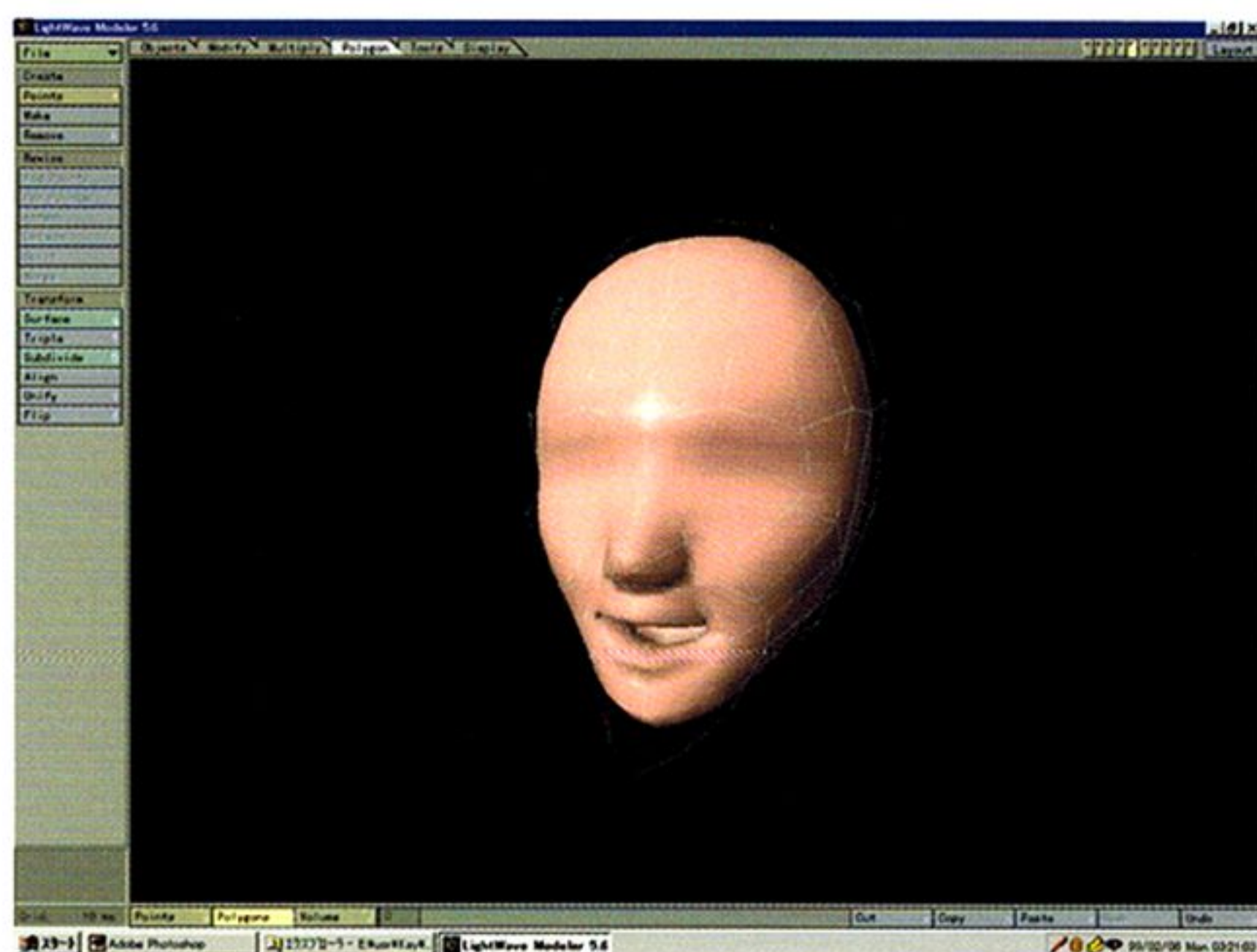



図7　同様にSmooth Shift押し出しで口を作ります。ポイントは，ただ制御ポリゴンを引っ込めるのではなく，この場合唇にあたる部分を一度盛り上げてから奥に引っ込めることです

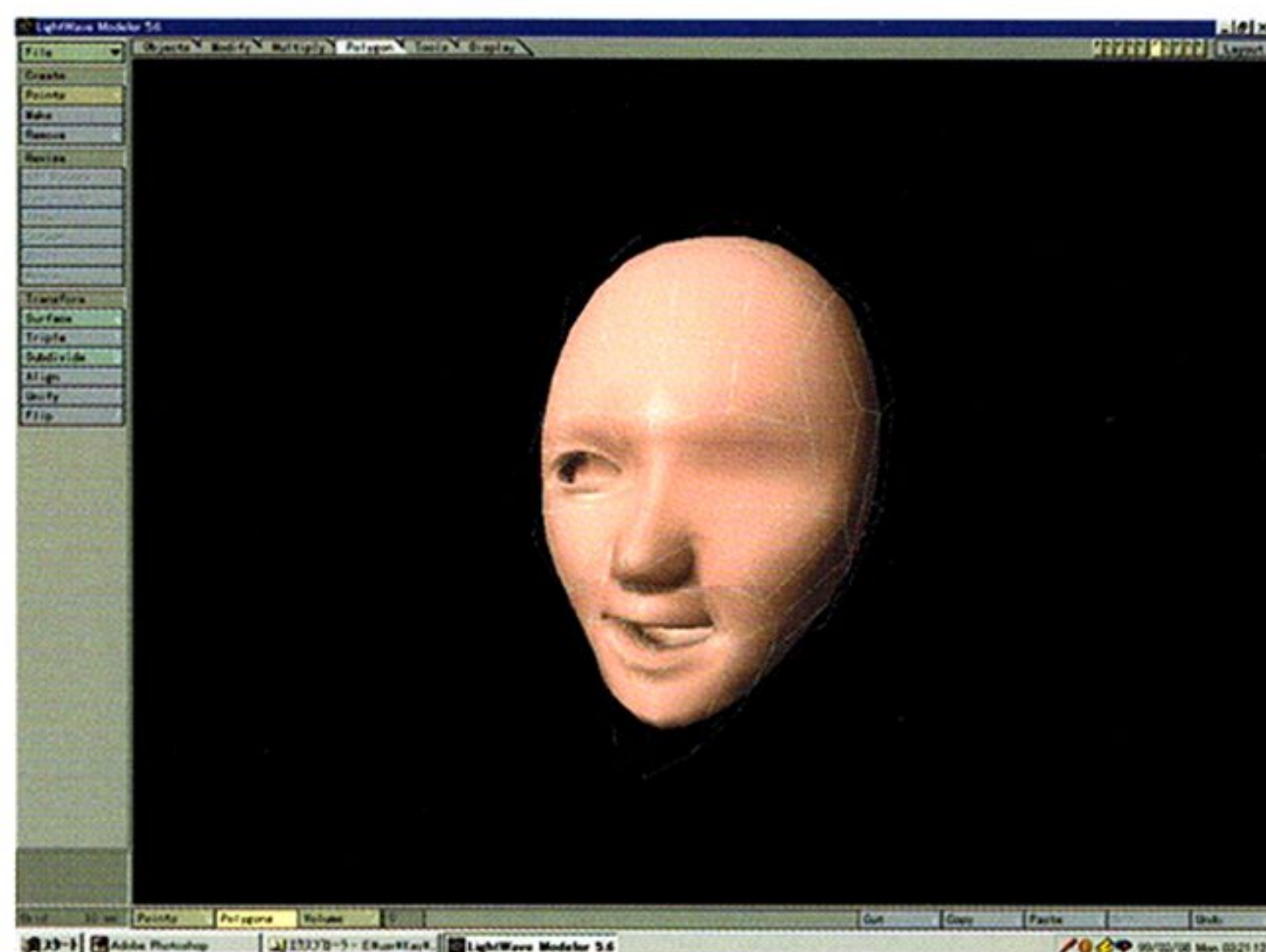



図8　口と同じ手法で眼も作れます。口や眼は，表情がついたときに周りの筋肉が大きく動くので，筋肉の流れを意識してモデリングしていきます

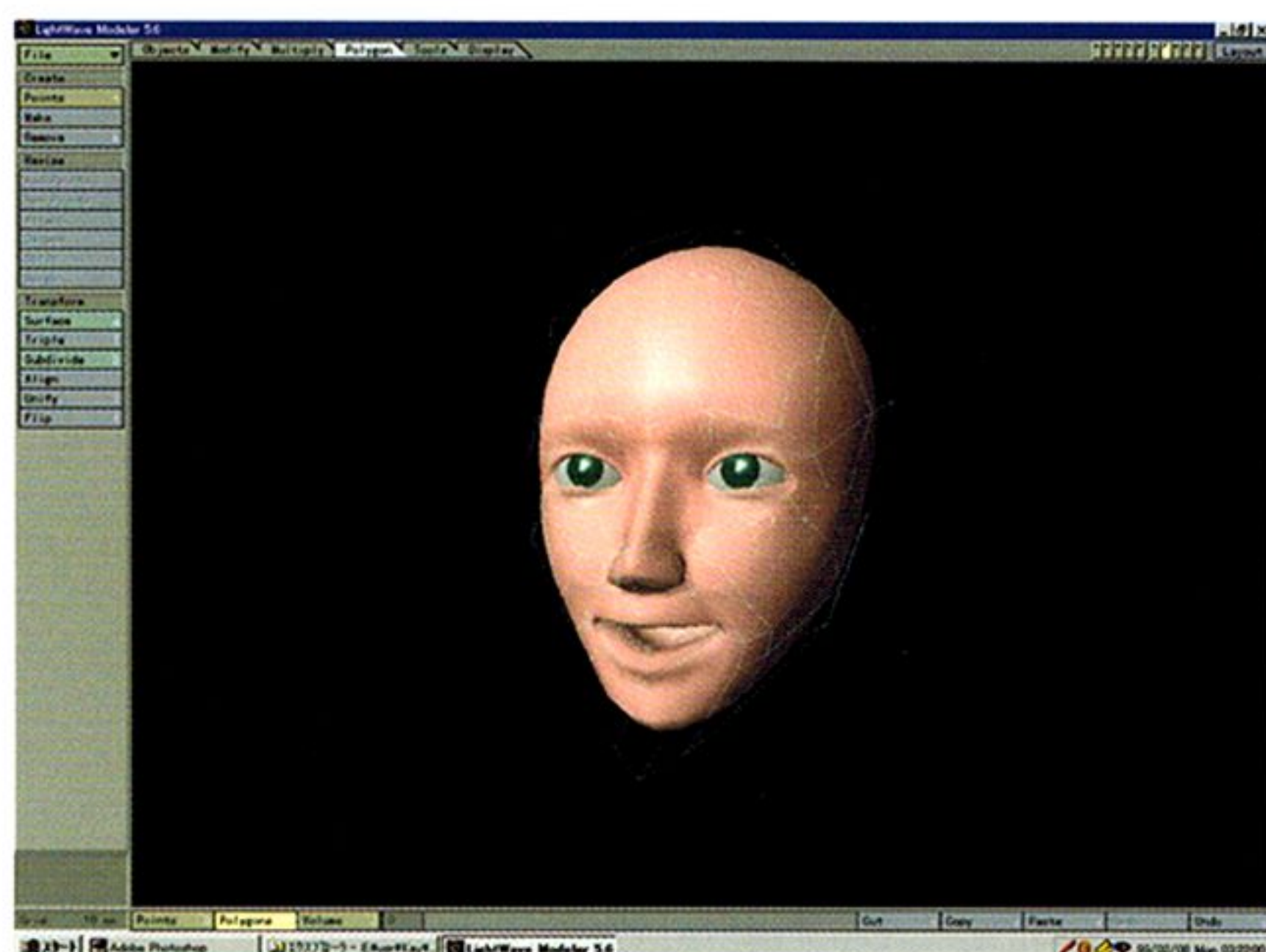



図9　ballツールで作った眼球をはめてみます。だいぶ顔らしくなってきた感じです。しかしこれ以上作り込むには制御ポリゴンが全体的に不足ぎみです

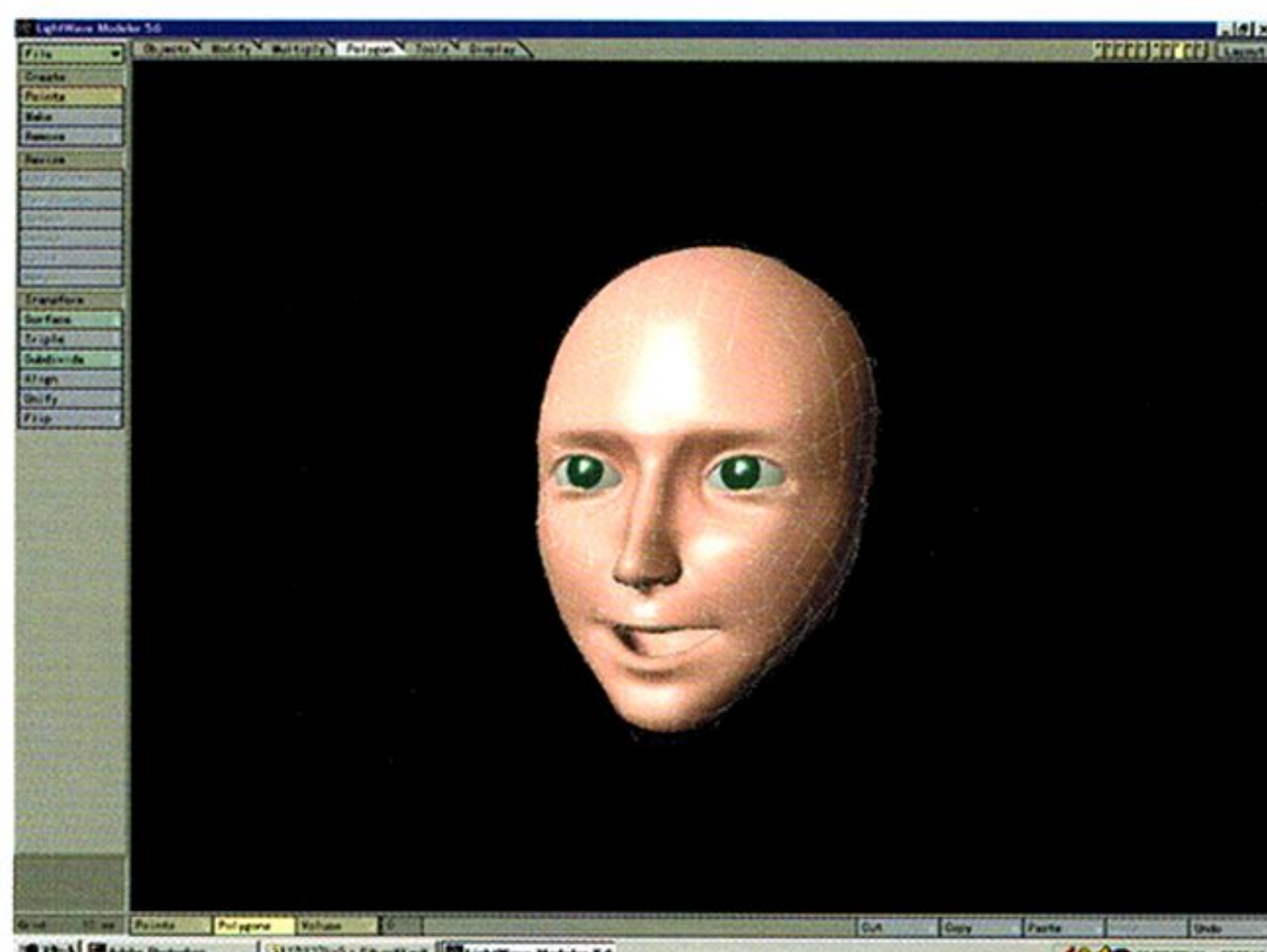



図10　ここで，さらに細部を作り込むため，Subdivideでラウンディング。制御ポリゴンは4倍になってしまいました(それでも制御ポリゴンは1000を超えていない)が，さらにきめ細かいモデリングが可能になります

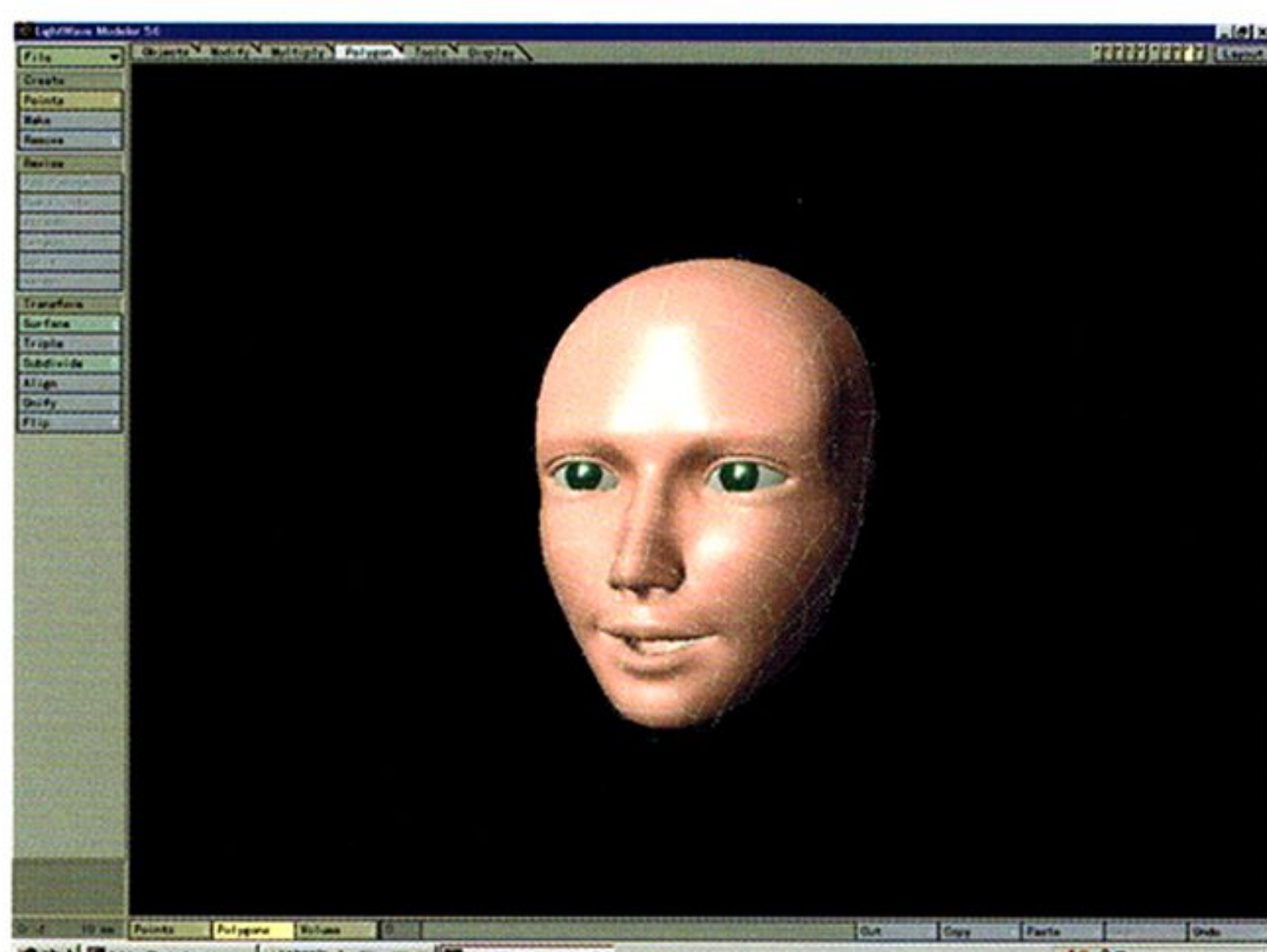



図11　増えた制御点をなるべく有効に使って造形を詰めていきます。この辺から，徐々にレイアウトでレンダリングしてチェックをしなければなりません。モデラー上で見るモデルと実際に撮影(レンダリング)したものとはだいぶ印象が異なる場合があるからです。ポイントは，35mmの標準レンズでレンダリングして，自然に見えるかどうかをチェックすることです

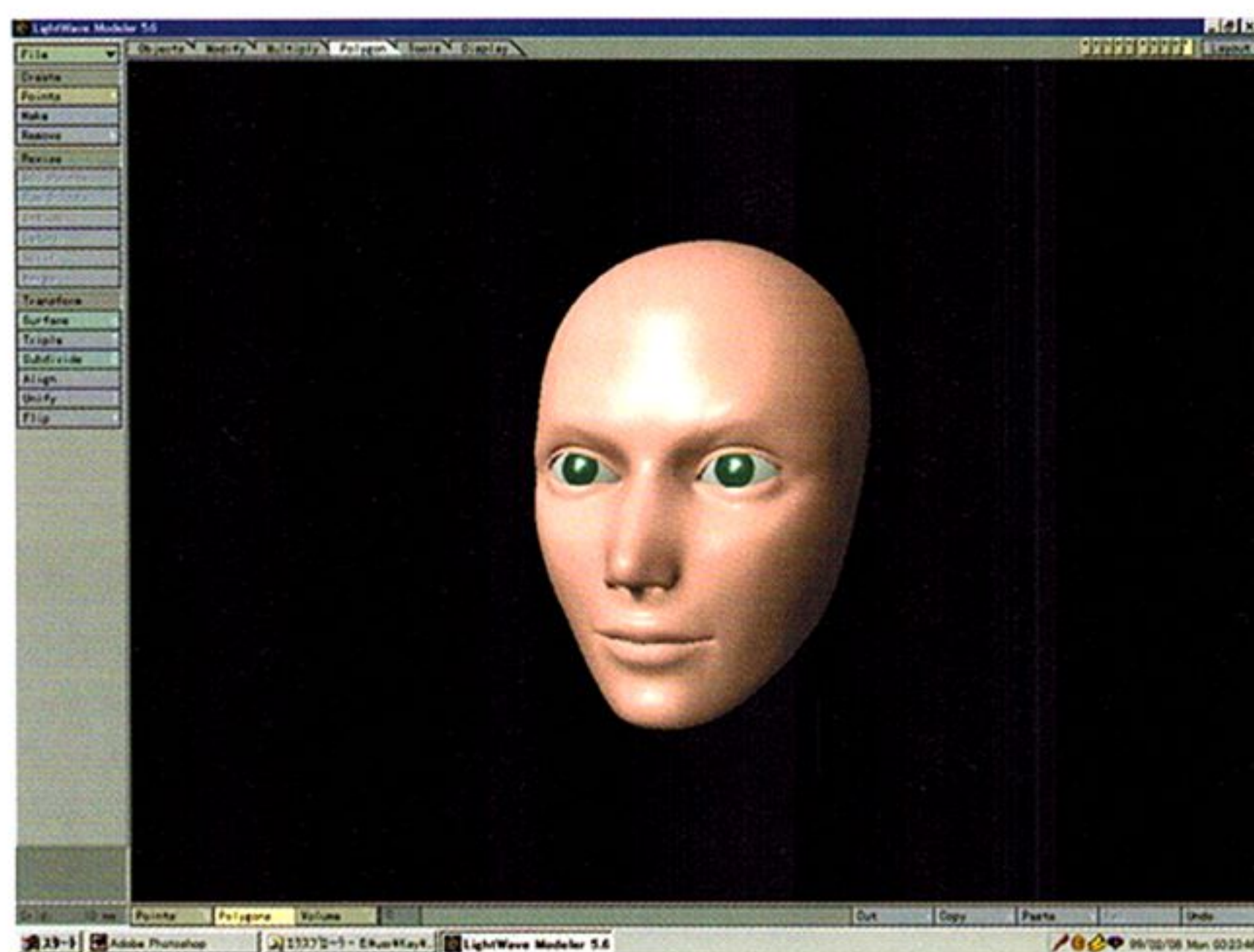



図12　完成した顔のモデル。キャラクターの個性や表情等を意識しながら細部の詰めをします。また，並行してテクスチャも制作し，レンダリングして仕上がりを確かめます。テクスチャによってモデルの印象はかなり変わるので，注意して制作しなければなりません

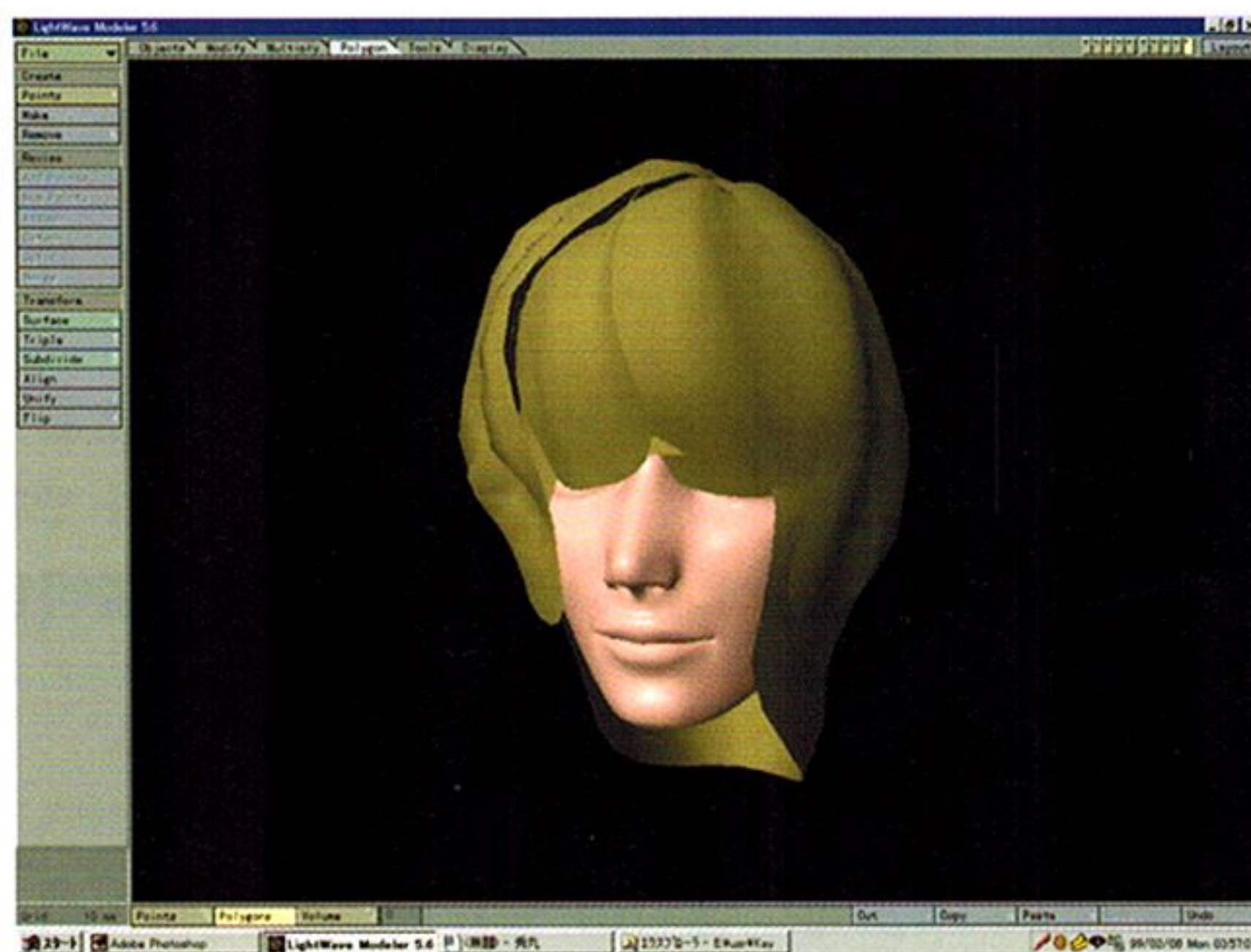



図13　髪の毛をつけた完成形。髪には透明度マップを施し，ボリュームを表現します。こうして，レンダリングしては，モデル，あるいはテクスチャを修正する作業を繰り返し，だんだんと完成度を上げていくのです

テクスチャ

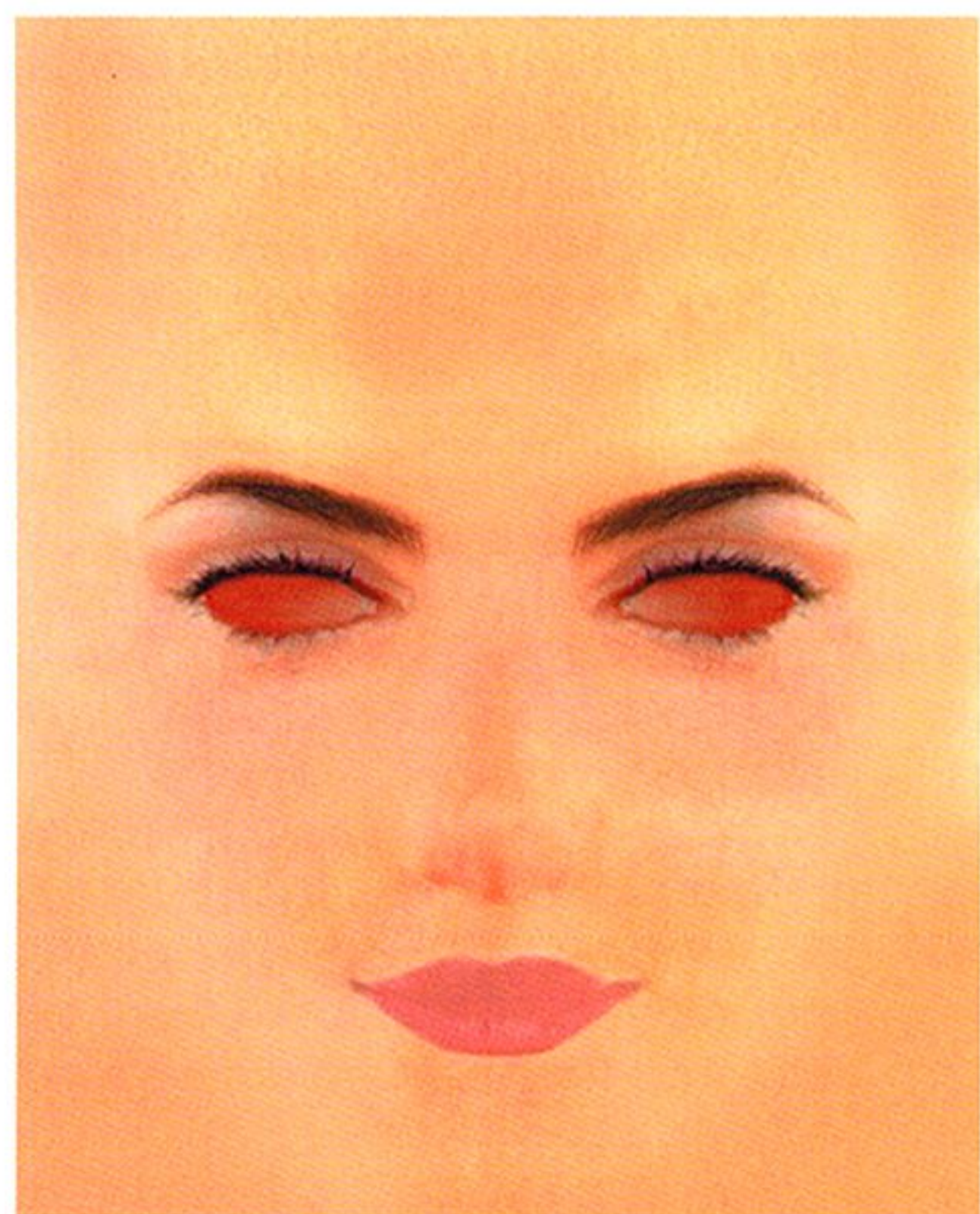



図14　顔のテクスチャ。リアルな表現にするために写真を参考に描きます。ポイントは色の情報だけでなく，スペキュラーやバンプなど，複数のテクスチャを用意して，それぞれのチャンネルに貼り込むことです

図15　髪の毛のカラーテクスチャ。髪の毛のボリューム感を出すのは非常に難しいのですが，テクスチャをうまく描けばそれらしく見せることができます

図16　髪の毛の透明度テクスチャ。ポイントは，髪の毛だからといって1本1本にこだわって描くのではなく，一房をボリュームとしてとらえ，流れを意識して描くことです

最後に

CGでリアルなキャラクターを作ることにどれだけの意味があるのかはわかりませんが，今回は習作のつもりでチャレンジしてみました。予想以上に大変だった部分もありましたが，石膏デッサンのようなもので，いろいろと勉強になる部分が多かったです。今度は男性モデルに挑戦してみたいですね。

THE CR☆SH DIVE
NAMAMUGI

筆者ホームページアドレス
(『THE CRASH DIVE』)
http://home.att.ne.jp/red/yosimizu/

第11回
アマチュアCGAコンテスト入選作品発表

今年もCGAコンテスト発表の季節となりました。お馴染みな人にはお馴染みなイベントですが，「なにそれ？」という方のために少し説明しておきましょう。

アマチュアCGAコンテストとは

アマチュアCGAコンテストは，〝アマチュアのCGアニメーション作品の発表の場を設け，広く一般にPRするとともに，その質的向上を促進する〟という主旨でProject team DoGAが開催しているコンテストです。1989年から毎年行っており，今年で第11回にあたります。

このコンテストはアマチュアCGA作家の活動成果発表の場として親しまれ，きわめてレベルの高い作品が数多く発表されるとともに，たくさんのCGA作家が生まれています。

使用機種や使用ソフトは一切問いません。また，過去にほかのコンテストで受賞した作品でも応募できます。毎年年末が応募締め切りで今年もコンテストが行われますので，CGアニメーションに興味のある方はProject team DoGAまで問い合わせてみてください。

●グランプリ　該当作品なし

●入賞作品

- **作品賞** *It's gauche to say.*　腰原 仁志
- **映像賞** *BATTLE CHASE 2*　河野 達也
- **ユーモア賞** *Mission : Li'l bit*　山口 秀行

●佳作作品

- *Humpty Dumpty*　杉山 摂朗
- *CGチュートリアルビデオ(マニア向け)Vol.1*　渡辺 哲也
- *ある暑い日に*　塩竈 信幸
- *DAIMAGINE*　DAVID T. SIA
- *存在/マンゴ・ミーニーズ*　嶺井 孝仁
- *せんたく危機一発*　清家 征雄

審査員

「ASAHIパソコン」編集長	五十嵐 文生
「日経CG」編集長	田島 進
「CG WORLD」編集長	永田 豊志
「デジクリ」編集長	村田 茂
「DOS/Vmagazine」副編集長	植木 章夫
アニメーション監督	渡部 高志
ファミリーほのぼの4コマ漫画家	寺島 令子
大阪芸術大学教授	吉田 幸一
PROJECT TEAM DoGA代表	鎌田 優

（順不同，敬称略）

本誌付録CD-ROMには，全入選作品のデータおよび入賞3作品の紹介ムービー（30秒程度のダイジェスト）を収録しております。

なお，入選作品発表会は下記の日程で行われます。どんなものかを知るのには，発表会に行くのがいちばんでしょう。

●東京会場

4月4日(日)開場12：30 開演13：00 終了17：00
中野区 なかのZERO 大ホール

●関西会場

4月11日(日)開場12：30 開演13：00 終了17：00
京都府民総合交流プラザ内
京都テルサ テルサホール

総評

●応募総数など

今年の応募総数は前回の2倍以上の296作品もありました。これほど急増した理由はよくわかりませんが，アマチュアCGAの世界が急速に広がっているのを感じます。

当コンテストでは，ビデオに収録可能な時間を入選作品数の基準としています。今年は昨年よりも20分収録時間を延ばしましたが，それでも応募作品が倍増しましたので，昨年ならば比較的上位にくるレベルの作品でないと今年は入選できないことになります。そのため，昨年入選された方の作品で，前作よりも確実にレベルアップしているにも関わらず，今年は選外になってしまうケースがありました。選外になった作品のなかにもアマチュアのCGA作品としてなんら恥ずかしくないものも多く見られ，そういった作品を選外にしなければならなかったのは非常に残念なことでした。

今年の傾向

不況の影響なのか，ノストラダムスだからなのか，今年はグロテスクで，やたらと人が死んだり世界が滅びるような，暗い作品が数多くありました。例年はロボットものとか宇宙戦闘ものが応募作の多くを占めていたのとは対照的です。それでも，結果として入選作品のなかで見ると，そのような作品はそれほど残っておりません。流行りものは似たような作品が多くなるため，評価が辛くなりやすいのでしょう。

応募者の傾向としては，CG系の専門学校から生徒の作品をまとめて応募してくるというケースが増え，全体の4割ぐらいを占めています。しかし，残念ながらそれらの多くは作品の形にまとまっておらず，入選にはほど遠いものでした。かなり高級な機材を用いているところが多いのですが，確かに作画品質は優れているものの，機材だけでは評価は高くなりません。やはり作品は中身で勝負といったところでしょう。

グランプリと入賞

残念ながら，今年もグランプリは〝該当作品なし〟となってしまいました。2年連続です。確かに飛び抜けた作品はありませんでした。しかし，これだけ全体のレベルが高くなると，飛び抜けることはほとんど不可能でしょう。ですから，今年は特に飛び抜けていなくても審査の最上位の作品にグランプリは出すつもりでした。

ところが，全審査員の得点を合計すると，上位2つの作品が，完全に同点になってしまったのです。完全に同じ得点なのに，片方をグランプリにするわけにもいきません。グランプリが2つというのも少し変です。準グランプリを2つという案もあったのですが，基本的に当コンテストに準グランプリというランクはないので，そんな例外を頻発するのも困ります。

そのような経緯で上位3作品をすべて入賞としました。というわけで，この入賞3作品のどれもが，例年のグランプリに相当するレベルだと思います。

●佳作

入賞3作品が，ほかの作品を明確に引き離していたのに対し，佳作と入選の境をどこに設けるかはかなり曖昧でした。今年はグランプリがないので，その分少し佳作を増やそうということで，6作品にしたという程度です。

入賞がほぼ全員の審査員から安定してよい点数を得るのに対して，佳作は一部には絶賛されるが，一部にはきわめて不評という傾向があります。ですから皆さんも，佳作6作品のいくつかは〝なんでこんなんが佳作になっているの？〟と首を傾げることでしょう。

これは，問題点もあるものの，ほかにはない長所もあるということでもあります。今後どのように活躍していくのかが特に注目されている人たちといえるでしょう。

入選作品集ビデオ申込方法

DoGAでは，入選作品を収録した以下のビデオを通信販売で配布しております（発送は日本国内に限ります）。

- **第11回アマチュアCGAコンテスト入選作品集（2,500円）**
  今回紹介した，第11回アマチュアCGAコンテストの入選作品全26本を収録したビデオ（VHS 140分，解説本（40ページ）付き）。
- **パーソナルCGアニメ The BEST（2,500円）**
  第10回までの入選作品の中から，特にリクエ

スト の多かった作品28本をまとめて収録したベスト版(VHS150分,解説本(40ページ)付き)。

・ロマのフ比嘉作品集(2,000円)

第9回でグランプリを受賞した比嘉さんの作品を集めたビデオ(VHS40分)。

申込方法

WWW,E-MAIL,ハガキ,郵便振替で受け付けております。詳しくはDoGAのホームページ
http://www.doga.co.jp/ptdoga/
をご覧ください。

・WWWでお申し込みになる場合

・DoGAのホームページ,
http://www.doga.co.jp/ptdoga/
のコンテストビデオ申し込みページからビデオを申し込むことができます。

・E-MAILでお申し込みになる場合

video-order-form@doga.co.jp にE-MAILを送ると,申込用フォームを折り返しお送りします(自動で処理しますので,本文は空でかまいません)。必要事項を記入のうえ,video-order@doga.co.jpに送っていただければ,ビデオ申込を受け付けいたします。

・ハガキでお申し込みになる場合

官製ハガキの裏面に,
・申し込み商品の内訳
(例:「11thコンテストビデオを1本申し込みます」)
・住所,氏名,電話番号
・DoGA 登録ナンバー(過去の申込者のみ)
を記入のうえ,
〒533-0032 大阪市 東淀川区 淡路 5-17-2-102
プロジェクトチームDoGA
「ビデオ発送」係
までお送りください。

・ビデオテープは代金引換郵便でお送りいたしますので,お受け取り時に代金をお支払いください。
・代金は商品の値段の合計+500円(代引手数料)になります(11thビデオのみの場合,2,500円+500円=3,000円)。
・お客様の都合により受け取りを拒否された場合でも,発送手数料(1,500円)を請求させていただきますのでご注意ください。

・郵便振替でお申し込みになる場合

郵便局で青色の郵便振替用紙をもらい,以下の必要事項を記入のうえ,代金をお振り込みください。

口座番号
00930-2-109598

金額
注文する商品の値段を合計した金額をお振り込みください。送料等は不要です。

加入者名
DOGA

払込人住所氏名欄
あなたの郵便番号・住所・氏名・電話番号

通信欄
1) 申し込み商品および数量
(例:「11th,ベスト,比嘉ビデオを各1本申し込みます」など)
2) 情報入手方法「ホームページを見て」
3) DoGA 登録ナンバー(過去の申込者のみ)

注意事項

・払込人住所氏名欄にこちらの住所を書かないようにしてください。
・キャッシュディスペンサの電信振替は使わないでください。電信振替では住所や電話番号がわかりません。
・記入漏れ,注意事項などを守られていない方の入金は,全額当チームへのカンパとして処理します。

なお,本誌付録CD-ROMに収録している入選作品紹介中のビデオ申込方法は,The BEST や比嘉作品集の情報が入っていないなど,いくつか内容が古くなっております。最新の情報は,DoGAのホームページ,
http://www.doga.co.jp/ptdoga/
で確認してください。

作品賞

It's gauche to say.

腰原 仁志

● 作品時間:10分30秒
● 制作人数:7人(CG:1,制作補佐:2,声:4)
● 制作日数:200日
● 使用機種
・AT互換機
・PC-9821Xa13
● 使用ソフト
・LightWave3D
・AfterEffects
・Premiere
・Photoshop

● 解説
姫に命を預ける兵士の前にマッドサイエンティストの陰謀が……。さぁ,変形だ! 合体だ! 前回の「STAY WITH ME」の世界観をベースに腰原ワールドが全開! ありがちな内容と思いきや,オリジナリティやテーマもしっかりした傑作。

映像賞

BATTLE CHASE 2

河野 達也

● 作品時間:17分40秒
● 制作人数:1人
● 制作日数:110日
● 使用機種
・AT互換機
・Mac互換機
● 使用ソフト
・LightWave3D
・Photoshop
・Premiere
・MediaStudioPro

● 解説
ある惑星の地下,閉鎖された空間で謎の巨大メカに襲われる。生きて出ることはできるのか? 18分間手に汗握る死闘が繰り広げられる。CGAは初挑戦だが,自主製作映画を作ってきただけあって,カメラワークなど,映像の基本技術は高い。

ユーモア賞

Mission : Li'l bit

山口 秀行

● 作品時間:2分42秒
● 制作人数:1人
● 制作日数:90日
● 使用機種
・AT互換機
● 使用ソフト
・AnimationMaster
・Premiere
・Acid

● 解説
2人に与えられた任務はスーパーコンピュータを盗み出すこと。盗んだらさっさと逃げればいいのに……。次から次へとテンポよく炸裂するギャグ! もうツッコミの入れまくり! ケチのつけようがない完成度で絶賛。

佳作

Humpty Dumpty

杉山 摂朗

- 作品時間：3分35秒
- 制作人数：1人
- 制作日数：180日
- 使用機種：・SGI O2・PowerMac G3
- 使用ソフト
  - Alias|wavefront Power Animator・Photoshop
  - Illustrator・AfterEffects

- 解説

ナチスのような独裁者が支配する世界。人々は連れ去られ、怪物のような姿に変えられていく……。どこを見てもプロとしか思えない完璧な作品。独特のグロテスクさが作者の個性として光っている。

CGチュートリアルビデオ（マニア向け）Vol.1

渡辺 哲也

- 作品時間：6分14秒
- 制作人数：1人（CG：1、音：1）
- 制作日数：135日
- 使用機種：AT互換機
- 使用ソフト
  - LightWave3D・AURA・Premiere

- 解説

昨年「リューセイバー」で話題を独占した作者から、CGを始める人へのメッセージ。ウチュートリアル刑事がCG界の裏側を暴く。すでに下手なプロより有名な作者だけあって、技術等の面ではいうことなし。あとは好きなだけ笑え！

ある暑い日に

塩竃 信幸

- 作品時間：5分32秒
- 制作人数：1人（声：2、音：1）
- 制作日数：180日
- 使用機種：Mac互換機
- 使用ソフト
  - AnimationMaster・StrataVideoShop・SoundWorks

- 解説

近未来、大学生の下宿に友人が訪れる。そこに1本の電話が……。審査員の評価が大きく分かれた問題作。あまりに斬新、従来では考えられなかった映像。これをCGAの新しい方向性として認めるべきか？　その審判は視聴者に委ねられる。

DAIMAGINE

DAVID T. SIA

- 作品時間：2分14秒
- 制作人数：1人
- 制作日数：65日
- 使用機種：AT互換機
- 使用ソフト
  - 3D Atelier・Premiere
  - Photoshop
- 解説

唐突に始まる戦闘シーンからは予想もできない展開に……。派手なだけでなく、ストーリー性やテーマ性もしっかりした秀作。当コンテスト初の海外入選者ということで、戦闘シーンひとつを見ても、日本人にはない感覚が新鮮だ。

存在/マンゴ・ミーニーズ

嶺井 孝仁

- 作品時間：2分6秒
- 制作人数：1人
- 制作日数：175日
- 使用機種：AT互換機
- 使用ソフト：
  - LightWave3D・Photoshop・MediaStudioVE

- 解説

架空のロックバンドのプロモーションビデオ。バンドのメンバーやサウナで汗を流す謎のおじさんたちといったキャラクターデザインがユニーク。モデリングなどのCGの基本レベルが高く、センスもよい。

せんたく危機一発

清家 征雄

- 作品時間：2分5秒
- 制作人数：1人
- 制作日数：110日
- 使用機種：Mac互換機
- 使用ソフト：
  - Infini-D・Premiere・AfterEffects

- 解説

コインランドリーにカンフーマニアの男と1匹の蚊。必然的に発生する戦い！　9th「BREAK RUNNER」の作者が放つバカムービー。Infini-Dでは世界トップクラスといわれる人体アクションも見もの。作者はこの作品のために半年間カンフーの修行をつんだとか。

入選

タイム・トラベル

吉澤 政昭

- 作品時間：3分50秒
- 制作人数：1人
- 制作日数：40日
- 使用機種
  - PowerMac 7500
- 使用ソフト
  - LightWave3D
  - AfterEffects
- 解説

時は21世紀、人類念願の夢だったタイムマシーンが完成され、1匹のお猿が過去に送り出された……。ユニークなキャラクターと予想できないオチ！　LightWave3Dのコンテストでも入賞している秀作。

オートマの虎

中林 圭

- 作品時間：5分10秒
- 制作人数2人
  （CG：1、音：1）
- 制作日数：120日
- 使用機種：PowerMacG3
- 使用ソフト
  - LightWave3D
  - Premiere・Photoshop
  - Bryce3D・AfterEffects
- 解説

荒野を走るサイドカーに乗る嫁と姑。些細なことから険悪なムードが……。昨年「荒野のからくり侍」で準グランプリを受賞した作者が、CG界で初めて嫁姑問題に挑戦する。あい変わらずの独自の作風を楽しんでください。

4アハウ3カン 自然の発動 真説編

西谷 和馬

- ●作品時間：2分50秒
- ●制作人数：1人
- ●制作日数：60日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・LightWave3D
  - ・Premiere
- ●解説

"4アハウ3カン"とは、古代マヤが予言する自然が人類に復讐を始める日のこと。この作品は、予告編です。当コンテストでは、予告編は未完成扱いですが、それでも選外にできなかったほどの映像センスは必見！

夢見回廊

尾崎 雅計

- ●作品時間：3分19秒
- ●制作人数：1人
- ●制作日数：90日
- ●使用機種：AT互換機
- ●使用ソフト
  - ・DOGA-L2
  - ・PhotoDeluxe
  - ・MediaStudioVE
  - ・PolyEdit・BURBSマクロ
  - ・DOGA-L2カスタマイズツール
- ●解説

湖の奥底から浮上した謎のカプセルの中には美しい少女が……。当コンテストには珍しい詩情あふれる美しい映像。ゆっくりとしたテンポが夢の世界へ誘う。

アリセイア

松本 由美

- ●作品時間：3分15秒
- ●制作人数：1人
- ●制作日数：50日
- ●使用機種：AT互換機
- ●使用ソフト
  - ・LightWave3D
  - ・Photoshop
- ●解説

片目を奪われ、とらわれの身の少女を救うため、単身ロボットに乗り込み……。ミュージッククリップのような軽快なテンポの映像。モデリングやモーションデザインなどCGの基本技術はレベルが高い。

受難のクリスマス

中尾 健次

- ●作品時間：1分43秒
- ●制作人数：1人
- ●制作日数：50日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・AnimationMaster
- ●解説

七面鳥の代わりとしてクリスマスに食べられるニワトリたち。一致団結して展開する一大作戦とは？　9thの「自由への逃走」と同様にアットホームな作品。奥様やお子さまも声で出演。

Second human EVAdam

小寺 知洋

- ●作品時間：8分45秒
- ●制作人数：1人
- ●制作日数：240日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・DOGA-L2
  - ・Premiere
- ●解説

ゴキブリのような人造生命体イヴァダムと機械を操るコンピュータとの壮絶な戦い。明日の世界を支配するのはどちらだ！　腕を吹き飛ばされても、頭がなくなっても活動を続けるイヴァダムの死闘は、当コンテスト前代未聞のものすごい迫力。

arx

ヤマダヒロユキ

- ●作品時間：7秒
- ●制作人数：1人
- ●制作日数：4.5日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・LightWave3D
  - ・Photoshop
- ●解説

昨年は「MUSE」で映像賞を受賞。今回は本編がたったの7秒（モデリング3日、モーションデザイン1日）。それでもちゃんと入選するのはセンスの違いか。掲載画像からは予想もつかない展開に、思わず「こんなのアリ?」

THE PROFESSIONAL

前田 晋一
三浦 真比等

- ●作品時間：2分50秒
- ●制作人数：2人
- ●制作日数：65日
- ●使用機種：AT互換機
- ●使用ソフト
  - ・LightWave3D
  - ・Premiere
  - ・Photoshop
- ●解説

依頼は100%遂行するプロのヒットマンが、遠くのビルからライフルを覗く。そこには……。映画のような渋い映像からは予想もつかない展開に注目。

R2 THE ROCKET RACER

安田 兼盛

- ●作品時間：2分14秒
- ●制作人数：1人
- ●制作日数：70日
- ●使用機種
  - ・PowerMac 9600
- ●使用ソフト
  - ・LightWave3D
  - ・Photoshop
  - ・Premiere
- ●解説

夜の美しいビル街をすり抜ける、ゲーム「ワイプアウト」ばりのロケットレース。果たして勝利の女神は誰の手に……。よくできたメカデザインや安心して見ていられる慣れたカメラワーク。爽快な映像をお楽しみください。

IMPERFECT

山田 政孝

- ●作品時間：9分30秒
- ●制作人数：1人
- ●制作日数：70日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・LightWave3D
- ●解説

女科学者が蘇らせた巨神が1999年7月に大地に立つ。そのころ火星では……。緩急の効いた戦闘シーンが見もの。ラストの急展開はなにを暗示するのか？

Return

村上 寛光

- ●作品時間：4分11秒
- ●制作人数：1人
- ●制作日数：240日
- ●使用機種
  - ・AT互換機
  - ・PC-9821Xa13
- ●使用ソフト
  - ・Photoshop
  - ・AfterEffects・Premiere
- ●解説

立ち並ぶビルの中、存在の希薄な人々が目的もなく流されていく。なくしたもの、忘れてしまったものはなに？　2Dアニメーションでもなかなか難しいこの柔らかな絵や色あいが見どころ。自然へのノスタルジーあふれる現代のファンタジー。

MANHOLE

森田 健

- ●作品時間：15秒
- ●制作人数：1人
- ●使用機種
  - ・PC/AT互換機
- ●使用ソフト
  - ・3D STUDIO MAX R2.5
  - ・Adobe After Effects 3.1J
  - ・Adobe Premiere 5.0J
  - ・Adobe Photshop 3.5J&4.0J
- ●解説

ここまでやるか？　異様に精巧にモデリングされた女子校生たちに作者のスピリッツが宿る。たった十数秒の映像だが、10thの「ラブレター」や「萌え燃えシスターズ」と同様、マニア必見。もう美少女モノもひとつのジャンルか？

RESQ AMIGO

成田 真人

- ●作品時間：10分41秒
- ●制作人数：1人
- ●制作日数：100日
- ●使用機種
  - ・AT互換機・PC-9821V200
- ●使用ソフト
  - ・DOGA-L2・PolyEdit
  - ・DOGA-L2カスタマイズツール
  - ・Video maid
- ●解説

2つの惑星間でいま、友好条約が結ばれようとしていた。しかし、その親善大使が事故で遭難！　ストーリーがわかりにくいなどの大きな欠点を持ちながら、多くの熱烈なファンを生んだ怪作。次から次へと出てくるキャラクターがとにかく楽しい！

和の食卓

胡本 満久
岩田 和哉
並木 禎久

- ●作品時間：1分20秒
- ●制作人数：3人
- ●制作日数：40日
- ●使用機種：SGI Indy
- ●使用ソフト
  - ・SOFTIMAGE
  - ・Photoshop・Premiere
- ●解説

食卓に残ったタクワンはひとつ。ここに爺さんと婆さんの熾烈な戦いが始まる……。たかがタクワンのために繰り広げられる究極魔法(?)は爆笑！　プロ並みの制作環境だけあって、CGの技術は最高レベル。

社長ノ民話

アロアズフゥ

- ●作品時間：2分30秒
- ●制作人数：2人
- ●制作日数：5日
- ●使用機種
  - ・PowerMac 7300
- ●使用ソフト
  - ・Photoshop・Illustrator
  - ・Premiere
  - ・AfterEffects
  - ・アニメスタジオ
- ●解説

ライブ用のオープニング映像ということだが、タイトルや内容には意味不明な点が多い。しかしこの作品を見れば、作者はただ者でないことはわかる。実験映像的な面もあり、審査員を悩ませた問題作。

THE LAST MISSION

長瀬 大学

- ●作品時間：5分30秒
- ●制作人数：1人
- ●制作日数：140日
- ●使用機種
  - ・AT互換機
- ●使用ソフト
  - ・DOGA-L2
- ●解説

除隊間近に起こる惨劇。戦友か家族か？ 全編に流れるオルゴールが切ない。ありがちなバトルロボットもので、CGとしての表現力も乏しいという見方もある。だが、見る人が見れば、このカメラワークや構図などの映像センスに絶賛！

Flipper pinball stories #2 Plungerの変遷を見る

市川幹人 Ichikawa Mikito

前回はプレイヤーの意志で重力に逆らうことを可能にした仕掛け「フリッパー」を紹介した。

フリッパーはコインオペゲームの世界にプレイヤーの腕がよければ長続きするという概念を持ち込み，「機械vs人間」という図式を生んだ。最後には人は必ず負ける(ゲームオーバーになる)が，それまでの得点がより高い人がより優れたプレイヤーとなる。この「機械vs人間」という図式は非常に斬新だったようで，フリッパーのないピンボールマシンにフリッパーを取り付ける拡張キットが発売され，ディストリビュータは改造してフリッパーを取り付けていた。

さらに「プレイヤーの腕がよければ長続きする」という図式は一般の遊び(カードゲームをはじめとする卓上のゲームやスポーツ)にもほとんど見受けられない。これは，人を相手にする遊びの場合勝敗がつくとそこで終わるため，腕がよければ長続きするわけではないことも大きく起因する。

フリッパーは「プレイヤーの意志で重力に逆らえる仕掛け」という役割をより明確にするため，現在ではアウトホールやや上方に左右対になって存在し，落ちるボールを寸前(アウトホールから3〜5cmくらい上方に存在するので，まさに"寸前"といえる)で救うのだが，今回は最初にボールを打ち出す仕掛け，プランジャーについて書いてみたい。

ピンボールの先祖ともいえるバガテルのときにはプランジャーは存在せず，キューでボールを打っていた。

1870年，Montague Redgrave氏はプランジャーを発明した。このときにはまだSpring shooterと呼ばれており，まだプランジャーという名前が定着していないことを感じさせる。ビリヤードのキューがHand shooterなのに対する呼び方なのだろう。この頃すでにプランジャーの先端にはラバーがついていたが，これはキューについているタップからきた発想と思われる。

Montague Redgrave氏はさらにフィールドにピンを使うこと，得点の入る穴なども発明した。これらは現代のピンボール，パチンコでも使われている。

彼は1871年5月30日にピンボールの発明に関するいくつかの特許を申請し認可されている。19世紀にアミューズメントで特許出願が行われていた辺り，オリジナルアイデアに対する米国の人の考え方を垣間見ることができる。

こののち，半世紀以上にわたってプランジャーに画期的な変化はなかった。

プランジャーは大きな変化を見せなかったが半世紀の間にピンボールは大きく進歩した。スコアを表示するBackbox(バックボックス)，フィールド中，唯一プレイヤーの意志で動かせるフリッパー，Bumper(バンパー)，Slingshot(スリングショット)，複数人プレイなど，現在も残っている多くのアイデアは1871〜1956年に登場したものだ。

またネガティブな面では，N.Y.をはじめとする多くの州でピンボールが禁止されたことが挙げられる。

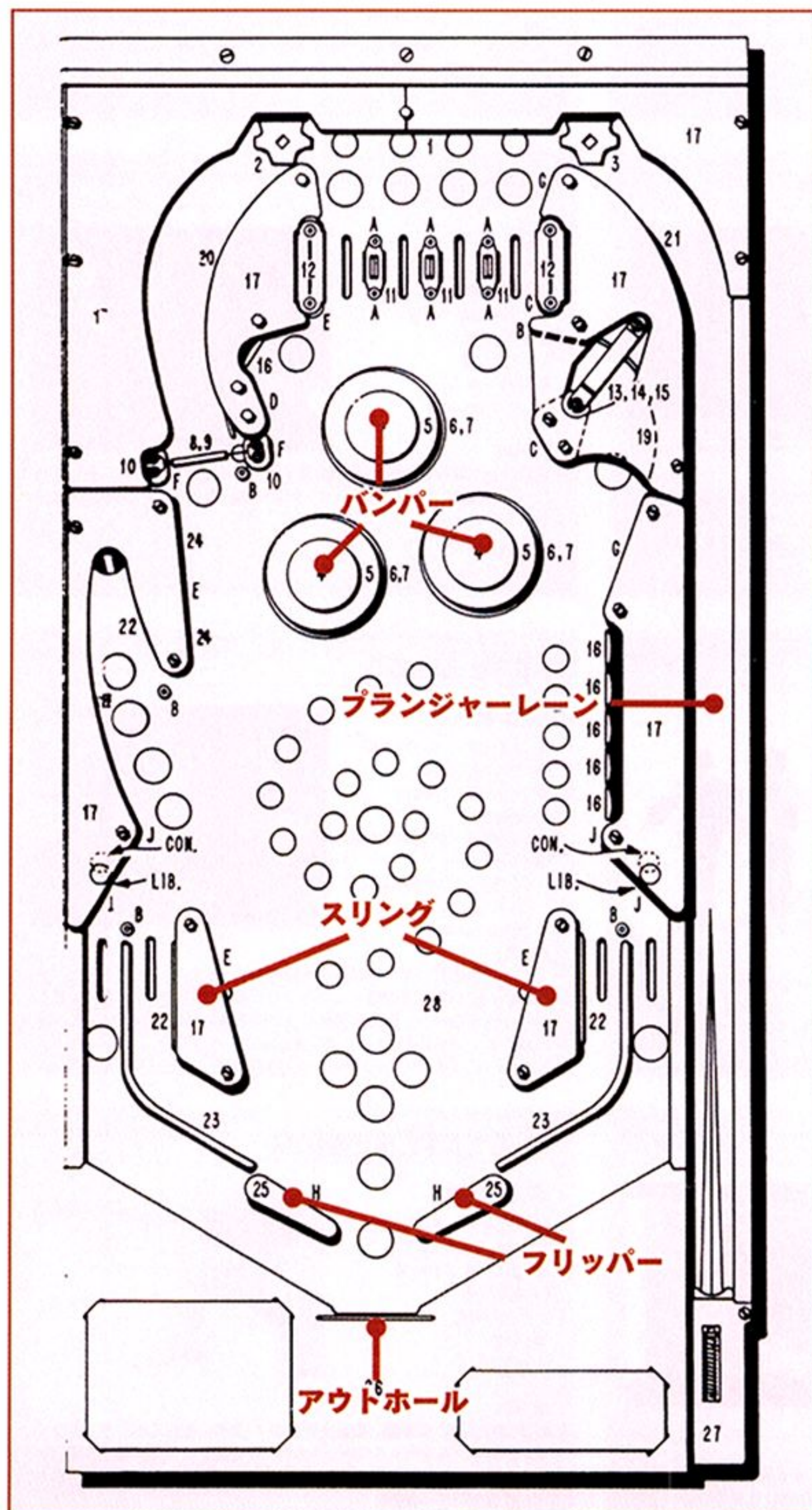



図1 ピンボール台のフィールドレイアウト例(The Shadow)

図2 普通のプランジャー。緑色のボタンはゲーム終了後，1コインで1ボール分だけ継続プレイするときのボタン

図3 プランジャークランク付き！ プランジャーの左下にあるのがクランクレバー。プレイヤーが操作するものがひとつ増えて少しお得な気分？

自動ボールフィーダー，オートプランジャー登場

●Balls-A-Poppin　Bally　56-8

1956年8月に登場したBalls-A-Poppin(Bally)でプランジャーは決定的な進歩を感じさせた。また，この作品は初めてMultiball(マルチボール)を採用していた。一度に複数のボールをフィールドに打ち出すためには自動的にボールを打ち出してくれるプランジャーが必要になり，Ballyはプランジャーに Solenoid(ソレノイド)を使用した。初のマルチボールをBallyはWildBallと名づけたが，子供たちにはCrazy Ballと呼ぶほど衝撃的だったようだ。しかし，1個のボールに神経を集中してじっくりと楽しむ傾向があった当時のピンボールのなかでは異色なアイデアを取り入れたこの作品は，当時ペイアウトに力を入れていたBally(この年Ballyはフリッパーピンボールを2作品しか発表していない。Williamsは10機種)の作品ということも災いしてか750台しか製造されておらず，マルチボールが日の目を見るにはまだまだ多くの歳月を必要とした。もちろんこの時点ではオートプランジャーとは呼ばれなかった。

またこの作品では，プランジャーレーンに自動的にボールを送り出すBall feeder(ボールフィーダ)を初めて搭載した。

現在のピンボールではゲームを開始するとプランジャーレーンに自動的にボールが出てくるがBalls-A-Poppinの発表までは手動でボールをプランジャーレーンに取り出していた。プランジャーのほかにクランクと呼ばれるレバーがあり，これを押すことによりボールをプランジャーレーンに取り出せる。

しかし，自動式のボールフィーダはすぐには普及せず多くのメーカーは60年代の中盤になってから採用を始めた。Balls-A-Poppinがヒットしなかったことも大きく起因していると思われる。

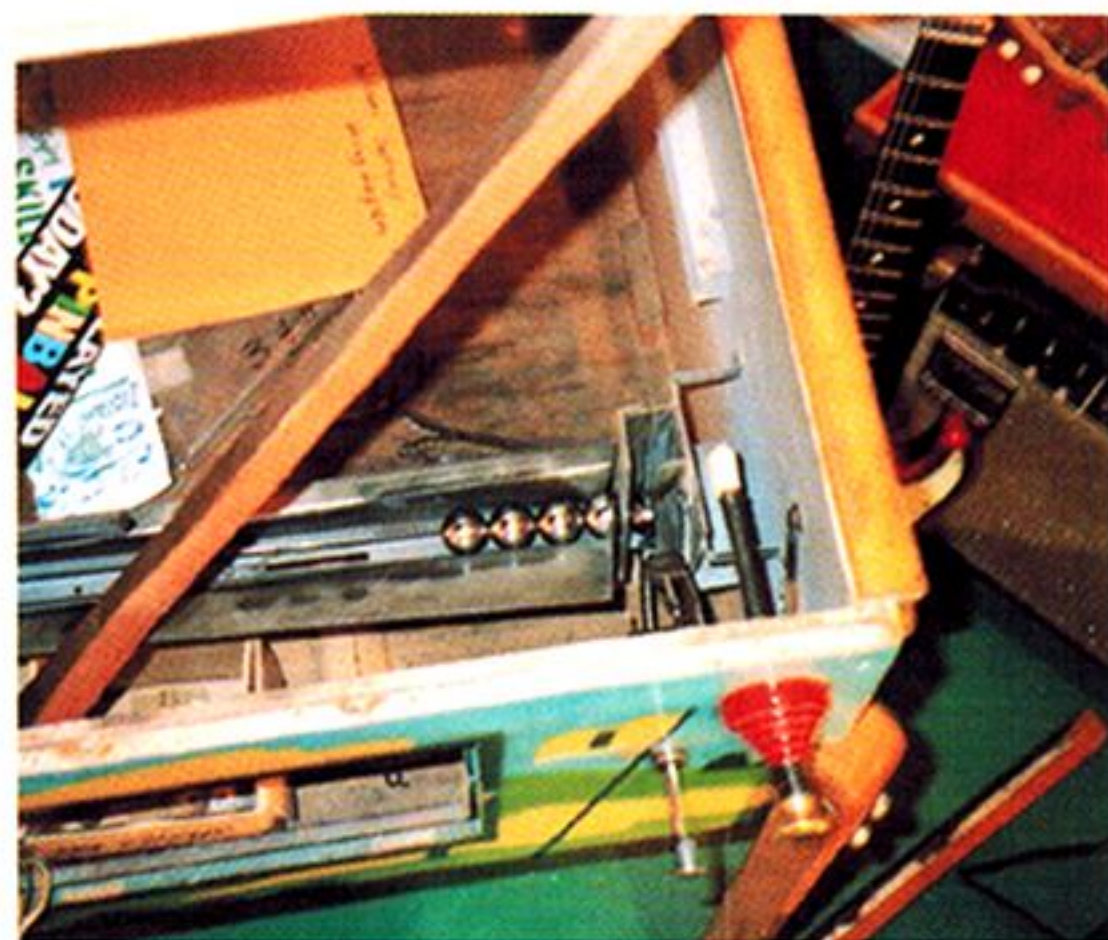



図6　手動でボールを引き上げるクランク。構造も原始的

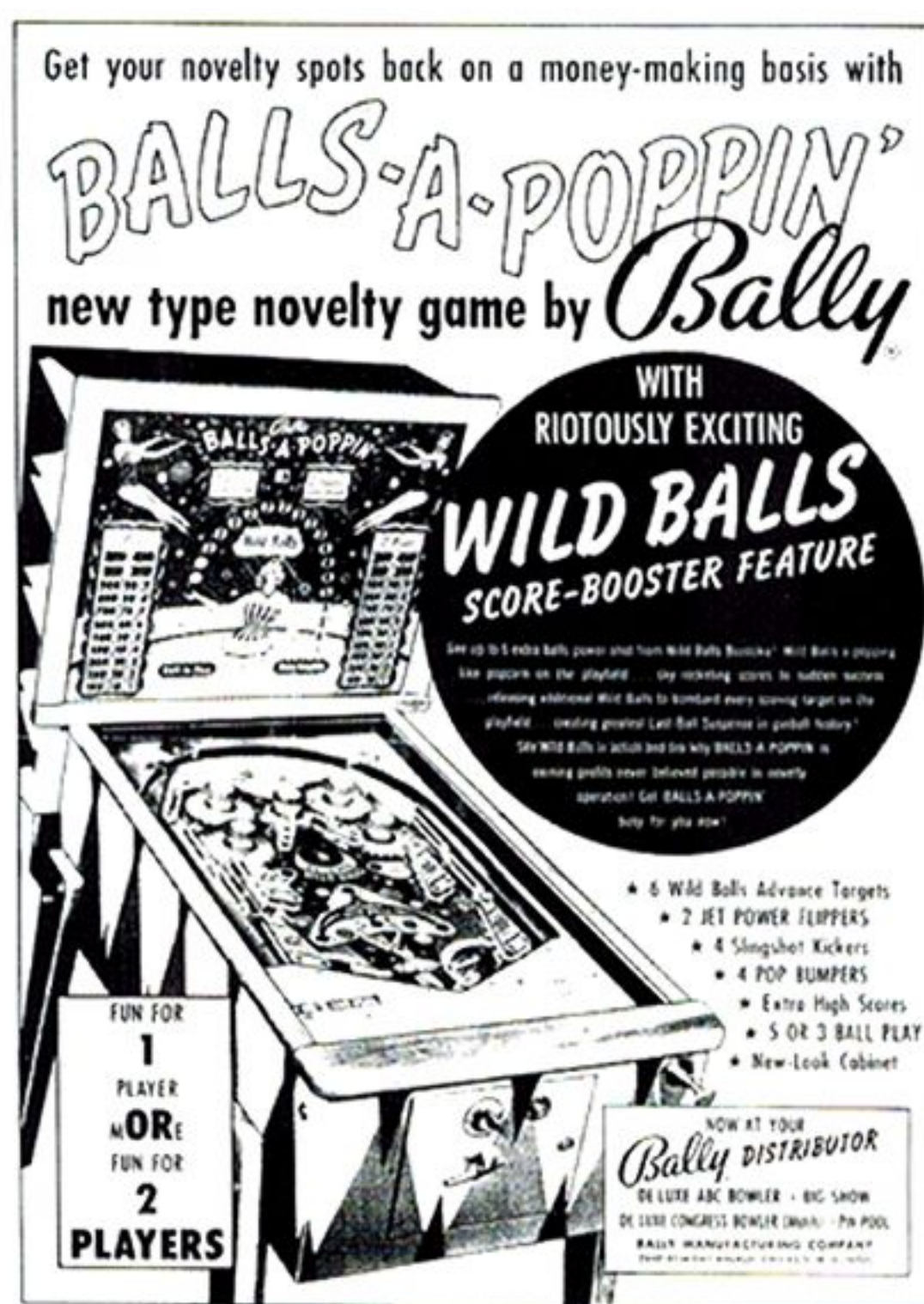



図4　Balls-A-Poppin

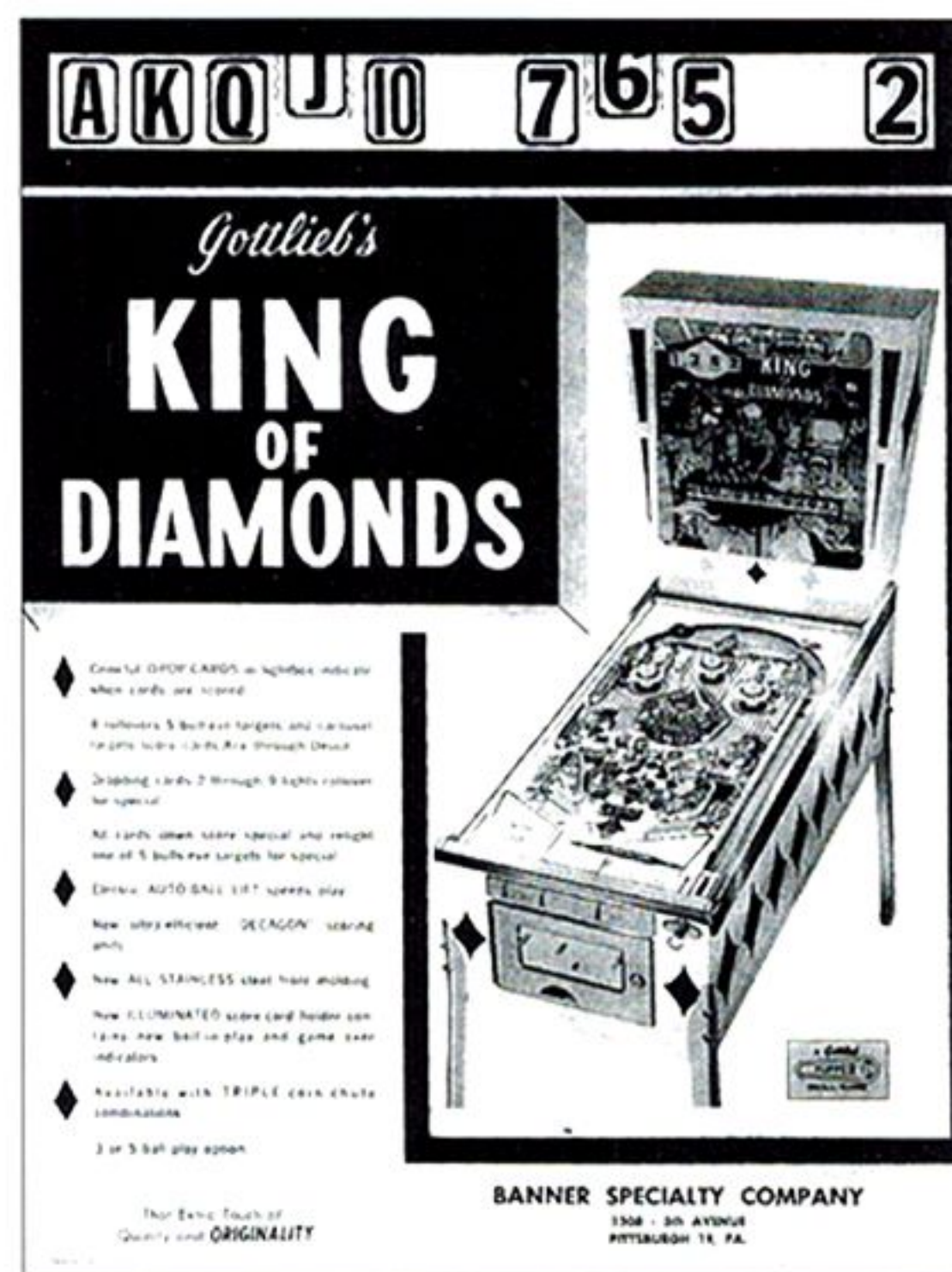



図5　King Of Diamonds。1967年Gottlieb作Ed Krynskyデザイン。上から5項目目に"Electric AUTO-BALL LIFT speeds play."と書いてあるところに時代を感じさせる

再びオートプランジャー登場

●Check point　Data East　91-3

Ball-A-Poppinから35年，オートプランジャーは復活する。

Data East Pinball Inc.が1991年3月に発表したCheck pointで再びオートプランジャーは復活する。この作品はオートプランジャーを復活させただけでなく，いまではすべての台に搭載されているDot matrix display(ドットマトリクスディスプレイ。この作品では128×16ドット)も搭載されヒット作となった。

'86年にHigh Speedが登場し大ヒットし，この作品で登場したマルチボール&ジャックポットはのちのすべてのピンボールに影響を与えた。マルチボールを始めるためにはボールをいくつかフィールドに固定することになる。そのため，ボールを落とすたびに交代して遊ぶ複数人プレイのときには，ほかのプレイヤーにロックしたボールを取られることがある。ロックしたボールを取られたプレイヤーは再びボールのロックから始める必要があったので不満を持つプレイヤーも多かった。

オートプランジャーの登場でボールのロックに関係なくマルチボールを開始することが可能になって，この不満は解決したが，この作品では「いつでもマルチボールを開始できる」メリットは複数人プレイのときにしか活かされなかった。

図7　Check point

ガンプランジャー登場

●T2　91-7　Lethal weapon 3　92-8

Data East Pinball Inc.に遅れること1シーズン，'89年にBally MIDWAYを買収したWilliamsもオートプランジャーおよびドットマトリクスディスプレイを搭載した作品を開発していた。大ヒットした同名の映画をテーマにしたT2である。T2は過去にもレーンチェンジやマグナバックなどを発明し，Hi Speedではのちのすべてのピンボールに決定的な影響を与えたマルチボール＆ジャックポットを搭載するなど，フィーチャーの特徴を最大限に引き出すことでは稀有の才能を持つSteve Richie氏のデザインである。

オートプランジャーをデジタルのボタンにし信頼性を高め，オートプランジャーであることを活かした，ボールロックに頼らないマルチボール，ガンのボタンをプランジャーのためのみでなく，ジャックポット獲得時にはフィールドにあるガンから球(弾？)を発射するためにも使うなど，オートプランジャー＆ボタンのメリットを最大限に活かしたテンポのよいゲーム展開の作品で，オートプランジャーの存在を決定づけ，ここでオートプランジャーの名称は定着した。

T2ではさらにドットマトリクスディスプレイでもデータイーストピンボール社の128×16ドットに対し128×32ドットのディスプレイを採用。フリッパーボタンを使って画面で遊ぶビデオモードもT2で初搭載し，ドットマトリクスディスプレイならではのフィーチャーを持ち込んだ。

T2はオートプランジャーを最大限に活かした素晴らしい作品であったが，このオートプランジャーはのちのピンボールに悪い影響も与えている。

マルチボールの主流は3ボールのマルチボールだが，オートプランジャーでない作品の場合，2つまたは3つのボールをロックしなければならない。つまり，マルチボールを遊ぶためには2回または3回連続のボールロックという条件を満たす必要があるのだ。これは初心者のプレイヤーには困難で，ゲームの華といえるフィーチャーを初心者には体験できなくしているといえる。一方オートプランジャーではフィールド上にボールがなくても次々にボールを発射可能なため，いつでもマルチボールが可能になった。ゲームのテンポもよくなり，初心者にもマルチボールを楽しめるのはメリットなのだが，いつでもマルチボールを始められる結果，デザイナーはルールを深く考える必要がないため，安易なルールの作品が多くなったことや，とりあえずマルチボールといった感じのインフレ的な作品を多くしてしまった。いわばデザイナーの逃げとしてマルチボールが使われてしまったのである。

また，オートプランジャーはボールを打ち出す強度は変えられないため，各ボール開始時のボールの行方がほぼ決まってしまうのもピンボールの重要な楽しみである「ボールをコントロールする」面を損なっている。

大ヒット作となったT2の登場から約1年，版権もののピンボールを多く発表していたデータイーストピンボール社もさっそくガンプランジャーのピンボールを発表。同名の映画を題材としたリーサルウェポン3である。ディプレイも128×32ドットを搭載。ガンプランジャーも搭載した作品だった。この作品までは新しいメーカーでありながらステレオサウンド，ソリッドステートフリッパー，ドットマトリクスなど，ピンボール業界に新たなアイデアを多数持ち込んでいたが，この作品以降データイースト社のピンボールは急激に独創性を失い，「テーマも版権ものならゲームの内容もパクリ！」となった。またこの頃から電源系を中心とする信頼性の低さも目立ち始め，米国のディストリビュータのなかには「3カ月の使い捨てピンボール」とまでいう人もいた。

日本ではSega Pinball Inc.の作品もData East Piball Inc.の作品もデータイーストが取り扱っていたが，'96年にピンボールからは撤退，'97年にはコインオペからも撤退した。国内でのピンボールの普及に大変力を入れていた。特に同社の制作したプレイングマニュアルはピンボールに馴染みのないプレイヤーにピンボールの魅力を伝え，国内での普及にも大きく貢献していた。また，かなりのシェアを誇って(世界的には15％に満たない)いただけに撤退は残念である。

図8　T2のプランジャー。ソレノイドを1個つけるだけなので中身は意外と単純

図9　T2。映画同様大ヒットした作品。17,000台を超える生産台数を誇る

図10-b　Lethal Weapon3のなかなか苦しい案内。制作は米国で行われていることもあり，販売業者の苦労が見受けられる

図10-a　Lethal Weapon3 T2と見比べてほしい。テーマも版権ものなら内容もパクリ？　と思わせるプレイフィールドである

オートとハンド！　2つのプランジャーレーン登場！

●Twilight Zone　Bally　93-6

図11　Twilight Zone

図12　Road Show

Banzai Run(Wil 88-9), Earth Shaker(Wil 89-5), Whirl Wind(90-1), The Addams Family(92-3)と次々にヒットを飛ばしたPat Lawlorの作品。彼の作品はいまのところすべてハンドプランジャーが装着され，毎回ハンドプランジャーのよさを活かしたスキルショットをプレイヤーは楽しむことができる。最新作のNo Good Gofers(98-1)ではオートプランジャーとハンドプランジャーの併用となっているが，Twilight Zoneでは，ハンドプランジャーとソレノイドによるボール打ち出し専用レーンを登場させ，それぞれのプランジャーのよさを得ている。また，氏のハンドプランジャーへのこだわりがうかがえ微笑ましさを感じさせてくれた。

彼はこの次の作品Road Showでは2つのプランジャーの搭載している。右側のプランジャーによるスキルショットにエクストラボールがかかりハンドプランジャーの扱いが得点に与える影響も大きい。

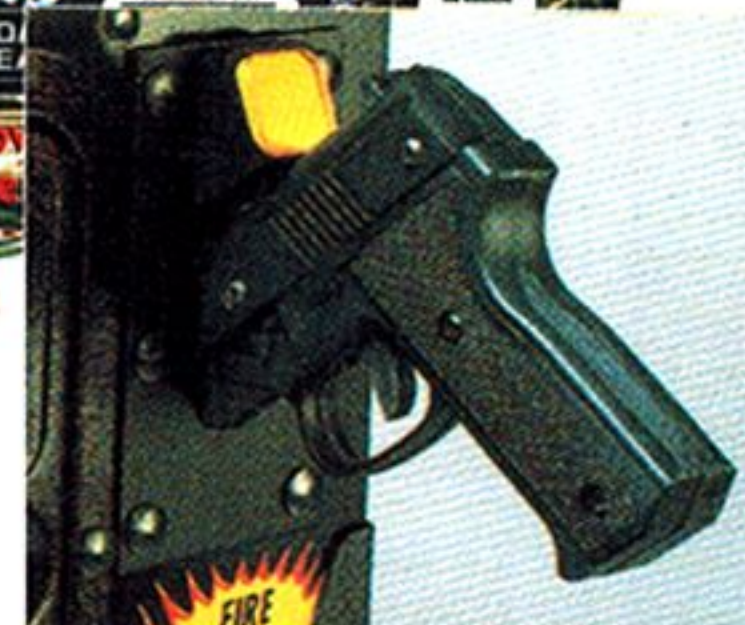
スマートミサイル登場

●Jurassic park　Data East　93-6
●Last Action Hero　Data East　93-8

ゲームの内容はBallyのThe Addams Familyのでき損ないだが，この2作品にはプランジャーに新たな試みがなされていた。すべての点灯しているフィーチャーを一度に獲得するスマートミサイルだ。マルチボールやエクストラボールなど重要なフィーチャーでも点灯していればいきなり獲得する，大変得した気分になれるフィーチャーだったが，どちらの作品も一度に数多くのフィーチャーが点灯することのない(複数フィーチャーは点灯するが)作品だったこともあり，あまり有効とはいえなかった。また，「1ゲームに一度だけ！」使用可能なフィーチャーで，戦略に絡める部分に乏しく残念である。

ストックの上限を1個とし，ゲーム開始時に1個与えられている形を取り，なにかのフィーチャーで再獲得可能にして多数フィーチャーが同時に点灯するルールにすればこれは定番となっただろう。このフィーチャーを活かしたピンボールを，コンピュータ用のピンボール(デジピンではない！)も含め誰かに作ってほしいと筆者は願っている。

図13　Jurassic park

図14　Last Action Hero

プランジャーボタンをボタンとしても活用

●Shadow　Bally　94-11　●Attack from Mars　Bally　95-12　●Medieval madness　Williams　97-6
●Champion pub　Bally　98-5　●Monster Bash　Williams　98-9

ガン式のオートプランジャーを除くと，プランジャーボタンにはライトがついている。フィールド上にライトが点灯している場合は「ここに打て！」の指示なのだが，プランジャーのライトをピンボール自体に役立てたのは意外と最近だ(ビデオモードでは'91年9月に発表のData East Pinball Inc.のStar Trekなど多数ある)。'95年末に発表されたBrian Eddyの作品Attack from Marsで登場。ゲーム中にフィーチャーを獲得していると点灯し，特定の場面でプランジャーボタンを押すとターゲットにボールを当てた扱いになる。Attack from Marsは1年近く米国のヒットチャートのトップを独占し，彼の次の作品Medieval Madnessでは14カ月にわたりトップを独占した。

ガン式プランジャーのShadowの時点で彼はこの発想をしたが，脚光を浴びるのはプランジャー自体にライトのついたボタン式のオートプランジャーだ。現在のWilliams/Ballyのボタン式のオートプランジャー搭載の作品では必ず使用されている。

ボールを打ち出したあとは，フリッパーボタンと台自体を揺らすことがゲームの操作となる。この発想はボールを打ち出したあとの操作にプランジャーボタンを加えた点で単純だが，ピンボールを大きく発展させる発想といえる。

今回はプランジャーについて書いてみた。やはりゲームの最初に行う動作なだけに，新たな発想が大きくジャンルを発展させる。今後プランジャーがどのように進化するのか，ピンボールに関わりながら見つめていきたい。

※　Data East Pinball Inc.社は日本のゲームメーカー，データイーストの子会社で米国シカゴに存在した。ときどき，日本で創られていると勘違いしている人がいるが，開発，製造ともに米国で行われているので注意！　現在はSega pinball,Incとなっているが同様に日本では開発，製造は行われていないので注意！

図15　Attack from Marsは米国では1年近くの間ヒットチャートのトップを快走したが，日本には数台しか輸入されなかった

図16　Medieval Madness
Attack From MarsのBrian Eddy氏の作品，14カ月の間ヒットチャートのトップだった。前作の焼き直しだが，前作の不満点を改良，新たなフィーチャーも多数加わり，この数年の作品のなかでも際立った完成度を誇る。安易な続編ばかりを作る日本のゲームメーカーにも「焼き直し」や「続編」について猛省してほしい

Notice
MIDWAY® is a registered trademark of Midway Manufacturing Company. THE SHADOW™ and BALLY® are trademarks used by Midway with permission. WILLIAMS, Lane-change and Multi-ball are trademarks of WILLIAMS Electronics Games, Inc. Entire contents of this manual ©1994 MIDWAY MANUFACTURING COMPANY, manufacturers of BALLY Amusement Games. All rights reserved.

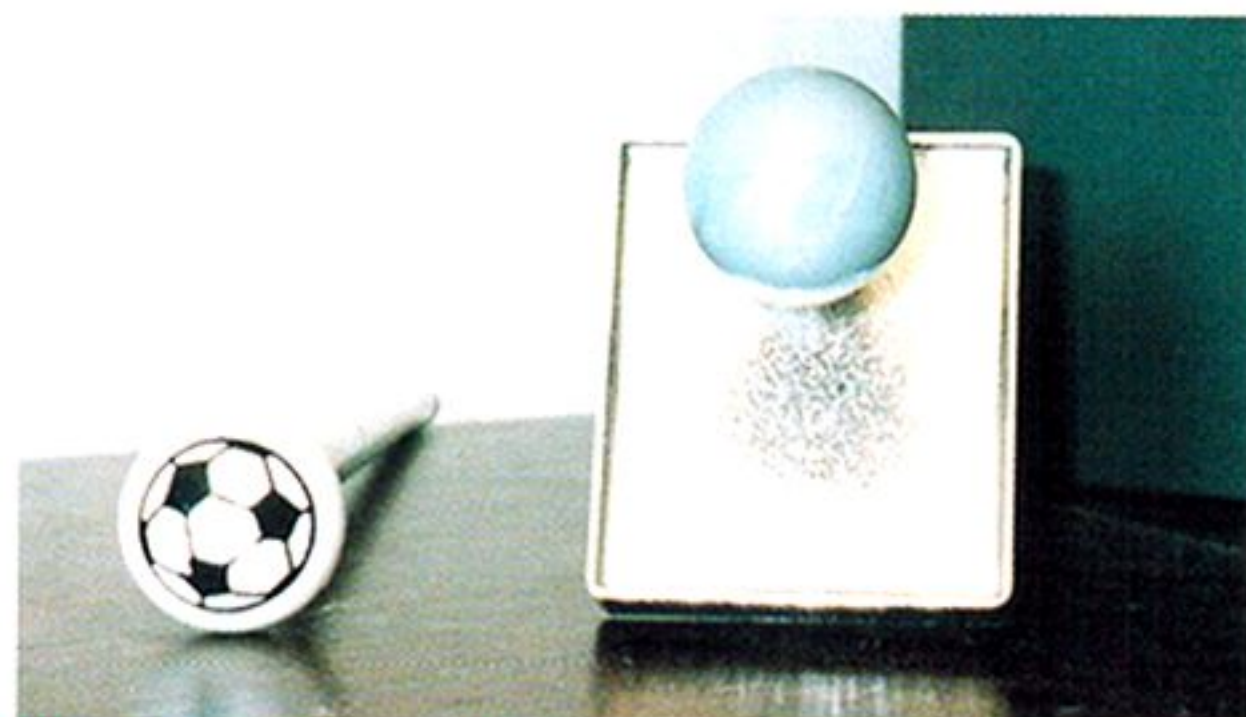
図17
コンピュータ用のピンボールでも実機でも多く見かけるマルチボール，これもまた登録商標。'94年頃まではWilliams/Bally以外のピンボールではTri-Ball，M-Ball，Mega modeなどマルチボールに違う名前をつけていた

さまざまなプランジャー

ピンボールには映画などの版権物の作品も数多く発表され現在も作られている。映画の主人公になった気分を味わえるようにとの配慮か，さまざまな形のプランジャーが存在し，プレイヤーを楽しませてくれる。

Apollo 13
ひねるとボールが出てくるが，どうやってボールを出すのか悩む人をたびたび見かける

NBA Fast Break
なんとプランジャーなし！　かと思ったらモールディングバーの真ん中にボタンがついている！

Star Wars
上げ下げ可能なうえにボタンがついている。一見上げ下げは意味なく感じられるが実は重要な意味が……

World Cup Soccer '94
コストの関係で没になったサッカーボールプランジャー

Pinball Action
ビデオゲームだがプレイした人も多いと思う。Lady Bug,Mr.DO,World Cup,Star Forceなどを手掛けた上田和敏氏の作品。アウトホールからボールが出てくるのが面白い

Scared Stiff
World Cup Soccer '94ではコストの関係で没になったはずだが……

マイクロソフトピンボールアーケード

市川幹人 Ichikawa Mikito

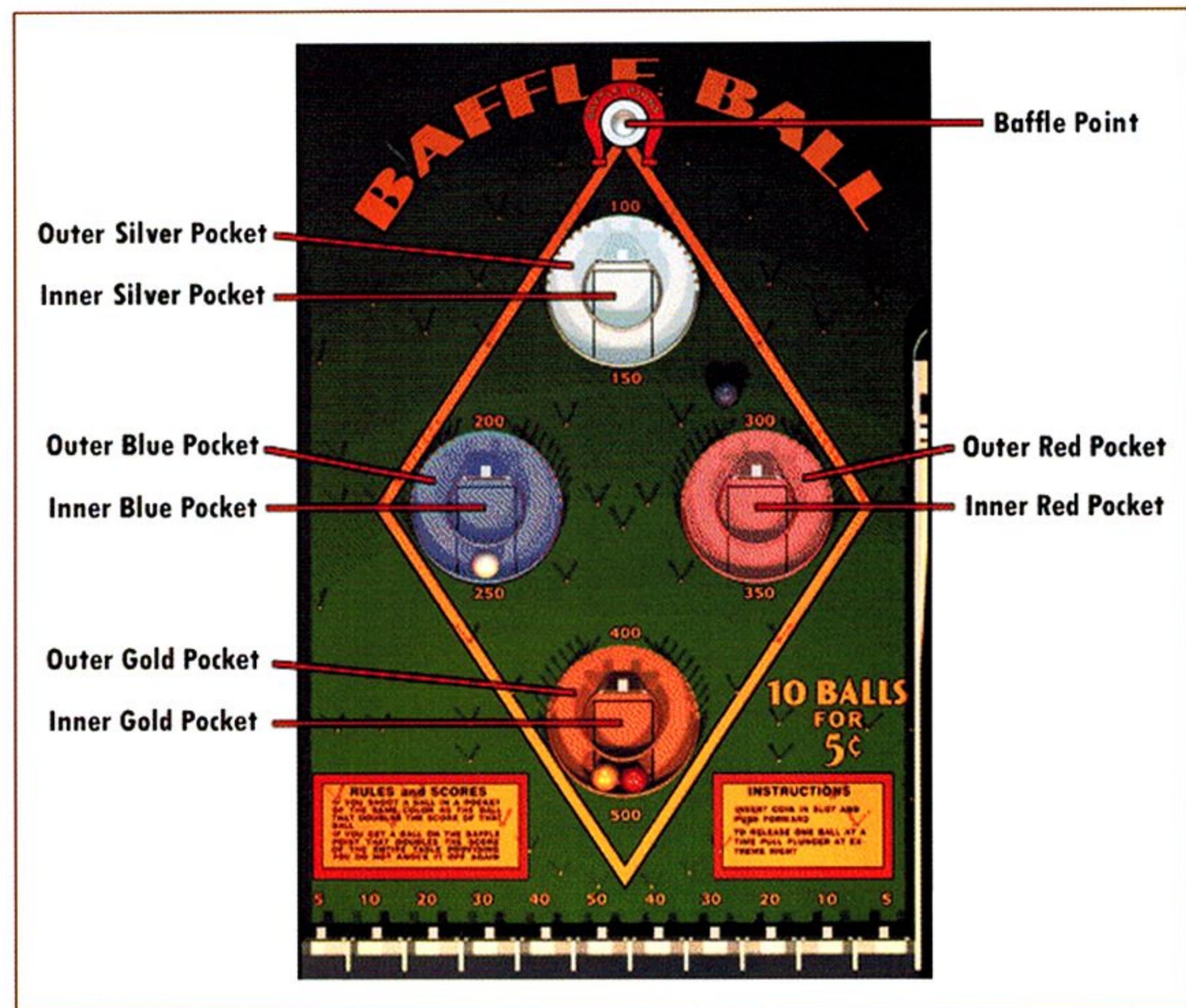
Planet ball。Baffle Ballのコピー。オリジナルは大変高価だがこちらは値段もお手頃

ピンボール産業に多くの貢献を果たし数々の名作を残したゴットリーブ。残念ながら1996年6月にこの会社は倒産してしまった。今回は同社の各時代のヒット作を収録した〝マイクロソフトピンボールアーケード〟のソフトウェア評価，原作の評価を記す。Flipper Pinball Story（特に前回の）とあわせて読むとさらに楽しめると思う。

ゲームの解説

Baffle Ball

Nov.1931
Design:David Gottlieb

Baffle Point
Outer Silver Pocket
Inner Silver Pocket
Outer Blue Pocket
Inner Blue Pocket
Outer Red Pocket
Inner Red Pocket
Outer Gold Pocket
Inner Gold Pocket

ゴットリーブ社のデビュー作品。多くの人にコリントゲームと呼ばれそうな，フリッパーのないピンボール，この作品の大ヒットがピンボール産業の礎を築いた！

年代が年代なだけに大変に古めかしいゲームで，まだ電気も使われてなくて，ひと目画面を見ると本当に「面白いのか？」と思ってしまうが，現在のピンボールには使われていない革新的なアイデアが投入されている！　特に以下に記す2つのアイデアはこの単純なゲームに戦略性，技術性を大きく加えている。

① ボールに色をつけ，同じ色のホールに入れると得点が倍！

このアイデアは現在のピンボールでも使われてない。使われてないというより，使えないのだ。

このゲームの原作にはスコアボードがないため，プレイヤーはゲーム終了またはゲーム中に自分で得点を数えるのだが，それが逆にこのアイデアを可能にしている。もし，自動的に複数色のボールを判定できるハードを開発すると，それだけでコストもかかり，信頼性を保つことも難しくなる。数多くのピンボールがこれまでに発表されてきたが，筆者の記憶だとフリッパーピンボールでこれに近いことを再現しているのはBallyのTwilight Zoneだけで，そのTwilight Zoneですら通常のボールとプラスチック製のボールの2種の識別のみだ。

② フィールド上部中央にあるBaffle Pointにボールが入っているあいだ得点2倍

このアイデアもまた，ほかの作品には見受けられない。

「ボールが入っているあいだ，倍の得点を獲得」ではなく。獲得済みの得点も含め倍にする。したがって，ボールが再び外へ出るとスコアはその場で半分になる。面白いのがBaffle Pointは浅いので揺らしすぎるとボールは落ちてしまうということだ。現在のピンボールには揺らしすぎに対する罰則としてTILTがあり，1ボールロストとなってプレイヤーにとってネガティブな状況になるが，TILT以外に揺らしすぎがプレイヤーにとってネガティブになるフィーチャーだ。このフィーチャーは技術的にも容易かつ安価に再現可能だが，フリッパーが登場する頃にはなくなってしまった。残念だ。

Baffle Ballの登場後に各社からピンボールは発表されたが，この2つのアイデアを採用した作品は意外なほどに少ない。Baffle Ballの成功を見て会社を興こしたBallyのデビュー作Ballywhooはボールは1種類で②のアイデアのホールも存在したものの，このホールは深くて，ゲームに作戦性を加えるには至らなかった。

ビデオゲーム黎明期にはコピーの問題があったが（最近ビデオゲーム界では公然と行われてい

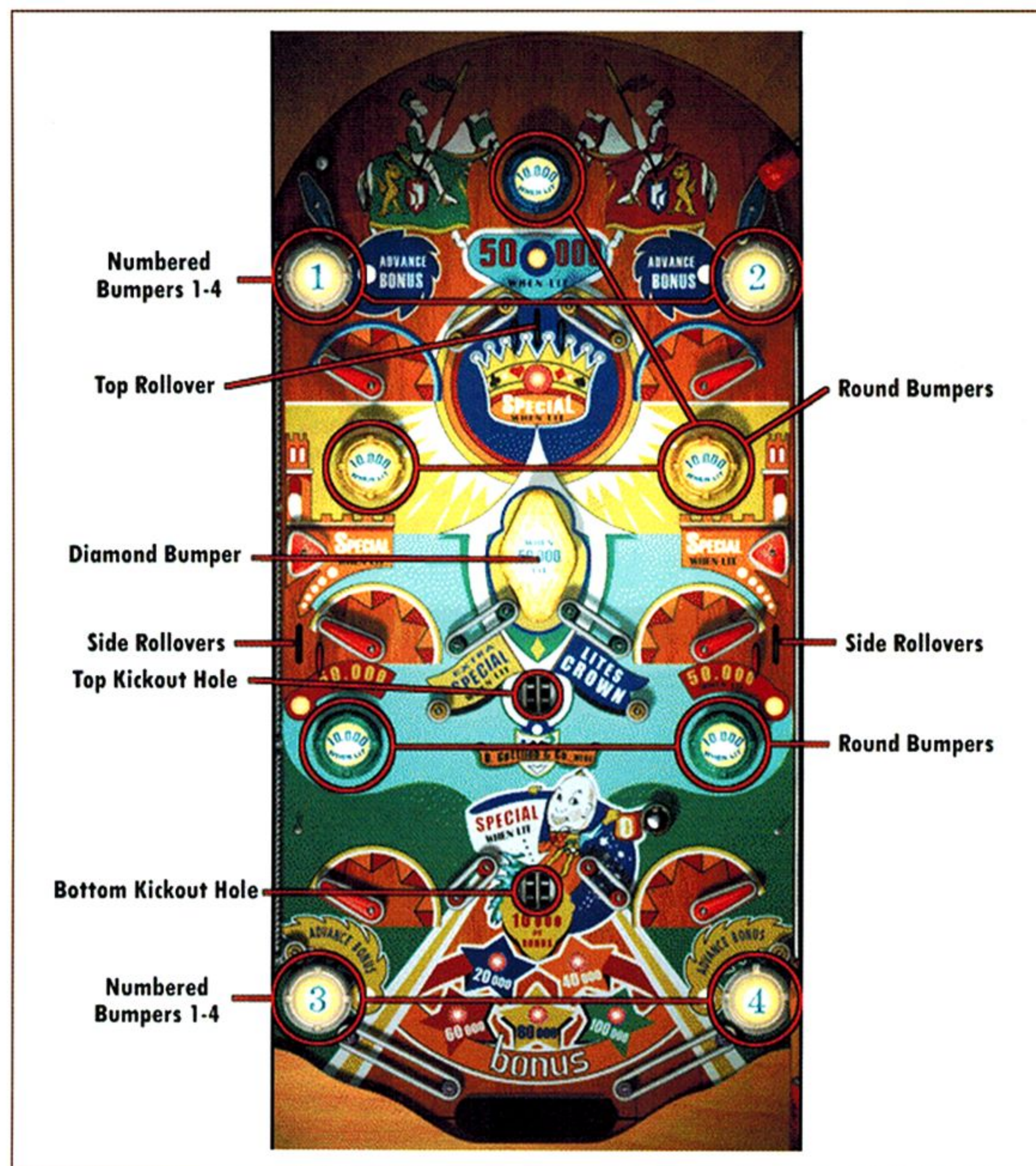
る？），Baffle Ballにもコピーは存在する。19世紀にすでに特許申請が行われていた半面，コピーも存在するのだ。米国の映画，音楽，ゲームなど娯楽産業は総じて権利関係に大変うるさいのだが，それもこのような歴史的背景があるからだろう。

David Gottlieb
Gottliebの創設者。デザイナーとしてではなく経営者として評価されるべき人。多くのメーカーがペイマシンを制作した時代に，スキルで遊ぶゲームを貫き通したことが結果的にコインオペ業界全体を作り上げたと評価できる。

サイクロン。多数のDiamond Bumperを搭載した作品

Humpty Dumpty

25 Oct.1947　Production Run:6500
Design:Harry Mabs　Art:Roy Parker

Numbered Bumpers 1-4
Top Rollover
Round Bumpers
Diamond Bumper
Side Rollovers
Side Rollovers
Top Kickout Hole
Round Bumpers
Bottom Kickout Hole
Numbered Bumpers 1-4

初めてフリッパーを搭載した歴史的作品。

ダイヤモンドバンパー，左右3対あるフリッパーが特徴的。似た形で翌年1月にはフリッパーをフィールド下方に配置したTriple Actionが発表された。フィールド下方にフリッパーを搭載することでフリッパーの持つ「重力に逆らえる」特徴が活かされ，瞬く間にHumpty Dumptyのようなフリッパー配置の作品はなくなった。また，Diamond Bumperも消滅するのは早かった。この仕掛けはあまりにもフィールドの多くに影響を与えるうえに，動的な要素がほとんどないからだろう。

フィーチャーとしてはBottom Kickout Holeにボールを入れるとボーナス得点が入るが，ボーナス得点獲得のあとに，必ずアウトホールにボールを落とされてしまう。このフィーチャーは現在のアウトホールボーナス（end-of-bonusともいう）に似ているが，アウトホールボーナスが制度化されたのは1970年11月にGottliebが発表したSnow Derbyが初めてである。この作品のようなフリッパー配置の作品がほとんど発表されなかったこともあり，このようなボーナス獲得形式の作品も非常に少なかった。

また，この頃の作品はスコアにドラムリールすら使用せずにライトで得点を表示している。クレジットのみ電球切れ対策かドラムリールを使用している。

Harry Mabs　90作品
1939年に発表されたHawthorneから1960年12月5日に発表されたMagic Clockまで90作品を開発。最後の約10年を除いてほとんどGottliebでの作品。ほとんどの作品がRoy Parkerとのコンビ。

Roy Parker　282作品
1936年3月に発表されたDaily Racesから1966年7月に発表されたHyde Parkまで約300作に関る。66年に没するまで現役のアーティストだった。Chicago coin，Gottlieb，Williams，Genco Mfg. Co.，Victory gamesと多数のメーカーの作品に関っている。

Knock Out

Dec. 1950:Production Run:3000
Design:Harry Mabs
Art:Roy Parker

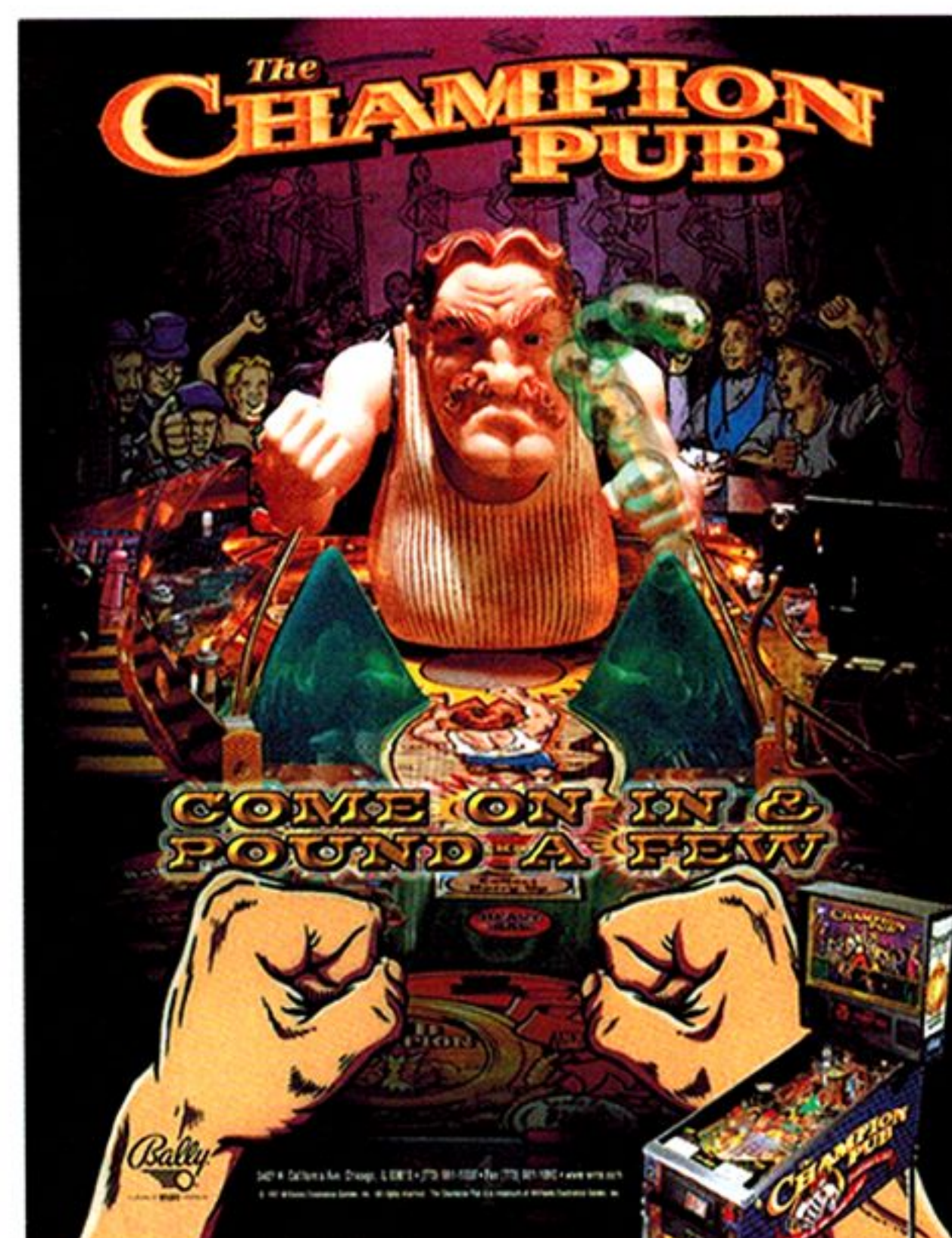
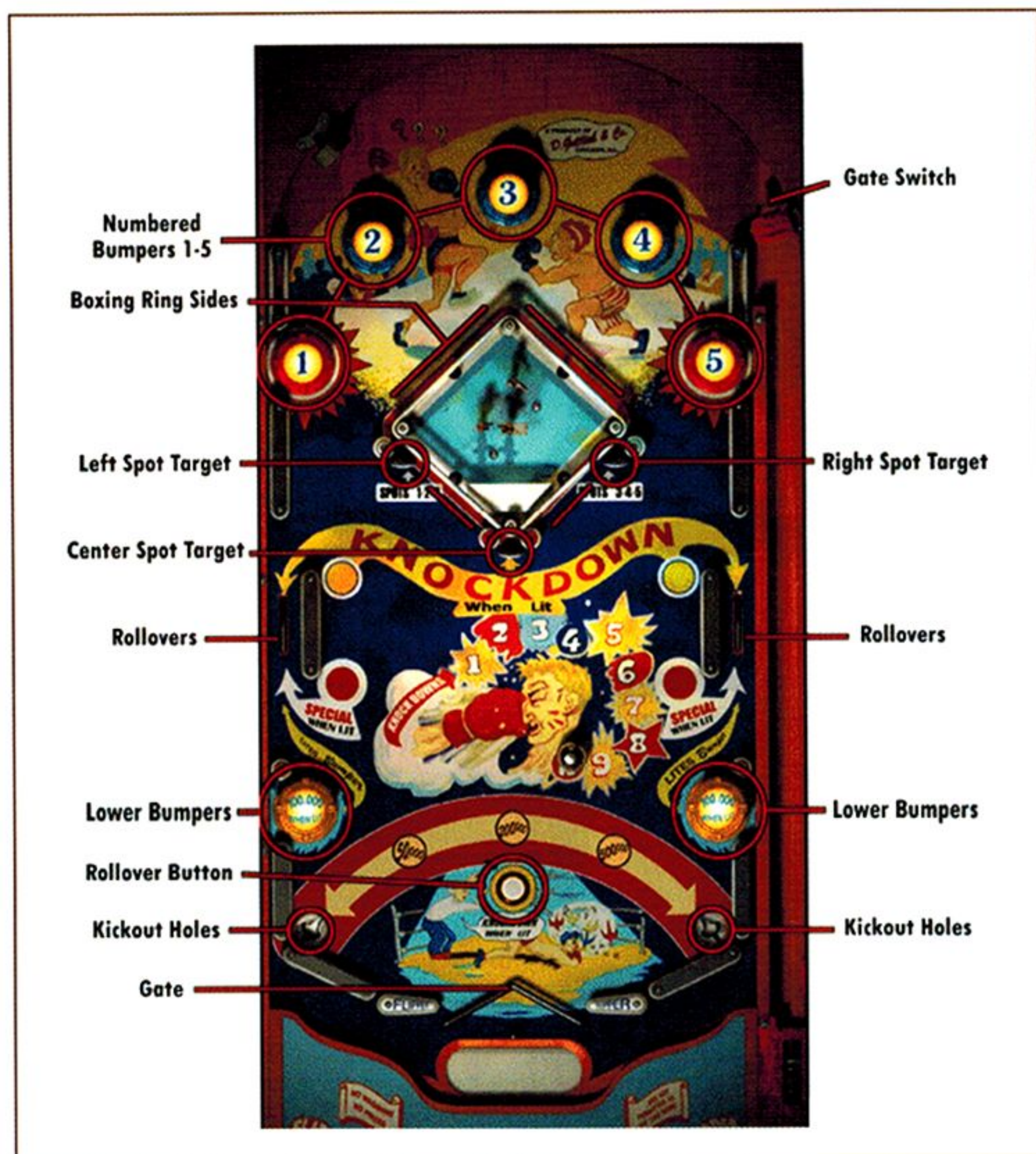
フリッパー近辺が個性的な作品のひとつ。Humpty Dumptyも同様だが，この'55年頃までの作品はフリッパー近辺が現代のピンボールとは大きく異なっている作品も多い。この作品のようにあまりにも2つのフリッパーが離れていると，2つのフリッパーを活用したテクニックが使用できないためか，'50年代中盤には完全に消えてしまった。

また，この作品にはフィールド中央上部にBoxing Ring Sidesが存在し，ボクサーが打ちあったり倒れたりするのだが，この年代の作品ではこのようなギミックは珍しい。現代のピンボールは立体構成を多用するようになったため，さらに凝ったギミックのものも登場している。

高得点のポイントとしては，

① フリッパーとフリッパーの間隔が大きく開いているのを見てあきらめないこと。意外なほどに長くプレイ可能
② Bumper 1-5に当てたりSpot Targetに当てて

Gate Switch
Numbered Bumpers 1-5
Boxing Ring Sides
Left Spot Target
Right Spot Target
Center Spot Target
Rollovers
Rollovers
Lower Bumpers
Lower Bumpers
Rollover Button
Kickout Holes
Kickout Holes
Gate

The CHAMPION PUB
COME ON IN & POUND A FEW

最近の立体ギミックの例。ボールをジャンプさせてぶつけるという構成だ

ダウンを回数多く取るよりも，Rollovers を揃えてKickoutHolesに何度もボール入れたほうが効率的
③ Lower Bumpersにボールが当たったときに右側のバンパーに当たった場合には右から，左側のバンパーに当たった場合には左から揺らさないとGateが下がったあとは危険なので注意！

Slick Chick

Apr. 1963　Production Run:4550
Design:Wayne Neyers　Art:Roy Parker

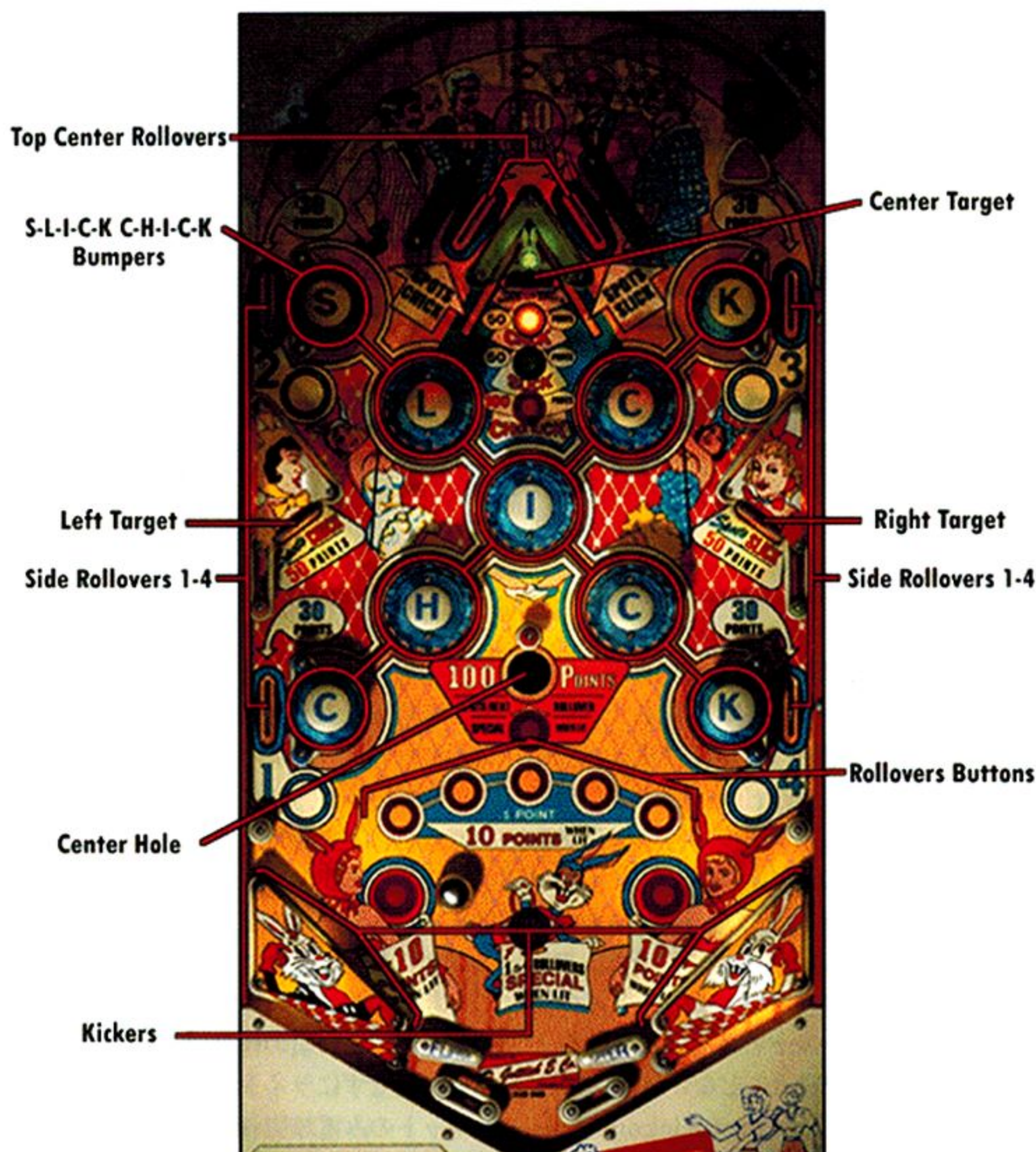
最近でも使用されているスペルを揃えるルールを搭載し始めた頃の作品。短いフリッパーを活かすため，フリッパーの下部にラバー張りの壁があることが球捌きに奥行きを加えている。

高得点を目指すにはとにかく「S-L-I-C-K C-H-I-C-K」の文字列を揃えること！　それには，

① Left Target, Center Target, Right Targetに当てる
② Top Center Rolloversを通過させる

Bumper近辺での揺らしでCenter Targetにボールを上手く当てられると高得点が見えてくる。現在のピンボールではスキルショットはどの作品にも搭載されているが，この頃はまだスキルショットとは呼ばれてはいないものの，プランジャーショットが大変重要になる。プランジャーでTop Center Rolloversを狙ってほしい。ゲーム開始時に5ボール与えられているうち，2ボール通過させることができれば得点はかなり伸びてくるはずだ。

Wayne Neyers　178作品
Gottliebに在籍したデザイナー。1949年8月に発表されたCollege Dazeから1976年3月に発表されたPioneerまで178作品を開発。Harry Mabs同様，Roy Parkerとのコンビが多い。

Top Center Rollovers
Center Target
S-L-I-C-K C-H-I-C-K Bumpers
Left Target
Right Target
Side Rollovers 1-4
Side Rollovers 1-4
Rollovers Buttons
Center Hole
Kickers

Spirit of '76

Dec. 1975
Production Run:10300
Design:Ed Krynsky, Wayne Neyers
Art:Gordon Morison

原作は大ヒットし，いまでも中古はオークション市場で高値のつく作品。'72年に現行の長さのフリッパーが登場。本ソフトウェア内ではこの作品からの台がロングフリッパーを採用している。

原作ではフィールド中央に存在するCenter Kickout Holeに入れるのが難しかったが，本ソフトウェアでは異様に入りやすくなっている。

Ed Krynsky
1965年のDodge Cityより1987年5月に発表されたAmazon Hunt IIまで200を超える作品を開発。すべての作品はGollieb から発表された。

Haunted House

Jun. 1982
Design:John Osborne
Art:Terry Doerzaph

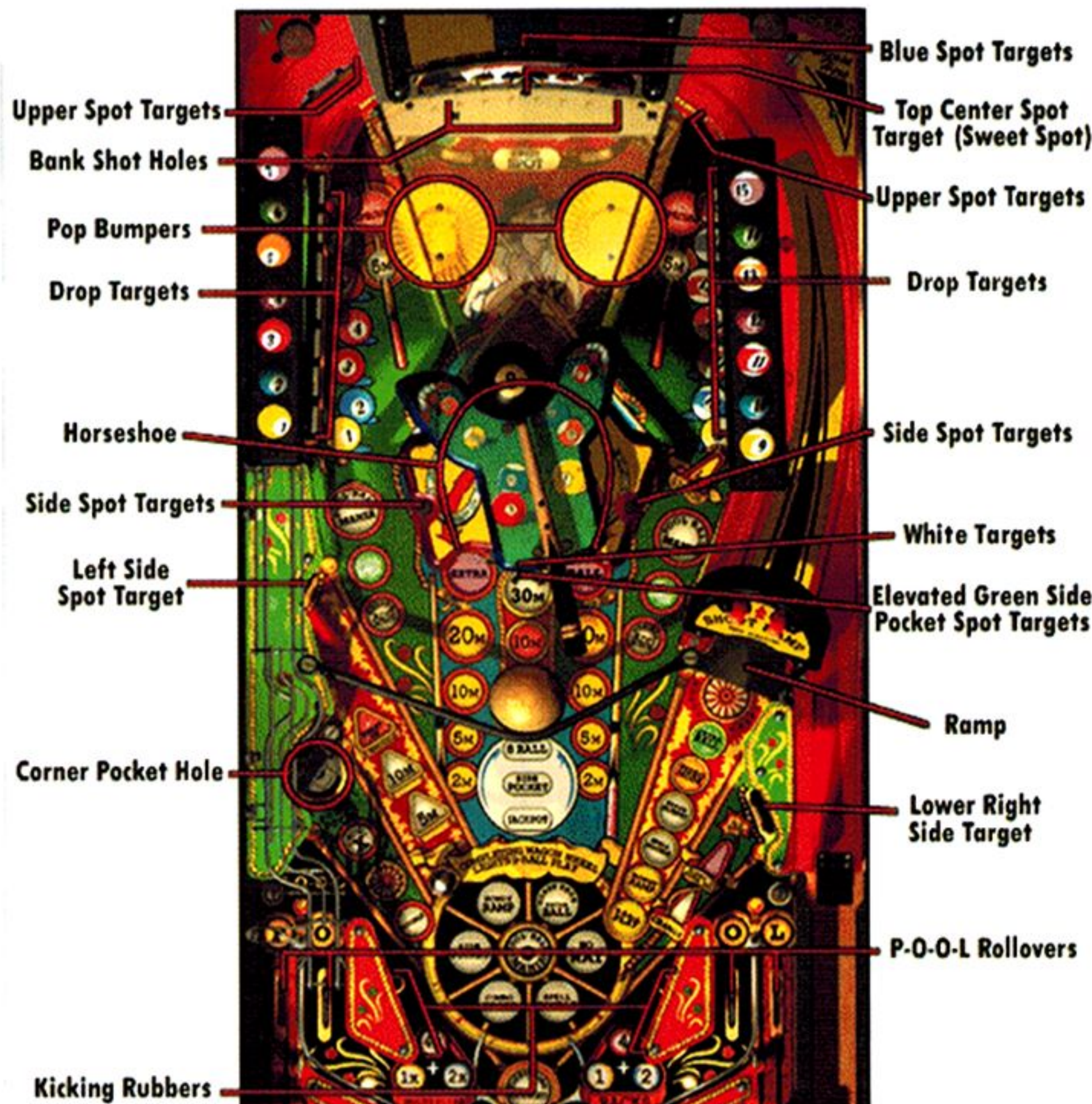
3階建てプレイフィールド，変形4枚フリッパーのフィールドのほか，さらに4枚のフリッパーのついたマシン。この頃のGottliebはプレイフィールドに個性的な作品が多く，これはその典型的な例だ。

Cue Ball Wizard

Oct. 1992
Design:Jon Norris
Art:Constantino Mitchell, Davit Moore

Blue Spot Targets
Upper Spot Targets
Top Center Spot Target (Sweet Spot)
Bank Shot Holes
Upper Spot Targets
Pop Bumpers
Drop Targets
Drop Targets
Horseshoe
Side Spot Targets
Side Spot Targets
White Targets
Left Side Spot Target
Elevated Green Side Pocket Spot Targets
Ramp
Corner Pocket Hole
Lower Right Side Target
P-O-O-L Rollovers
Kicking Rubbers

国内でもまだ多数見かける作品。'70～'80年代中期までのルールとマルチボール＆ジャックポット，ミニゲーム～ビッグゲームといった，ごく最近のピンボールのルール，派手なギミックがうまく噛みあった傑作。ド真ん中にある実物の手玉と8番ボールを見て一見「大味？」とか「クソゲー？」と思ってしまうが，つい百円玉を投入してしまった人も多いと思う。

Jon Norris
'80年代後半からGottlieb社の最期までを支えた。ギミックに頼らずに個性的なフィールドを描くことと，短い開発期間にクオリティの高いフィールドを制作できる希有な才能を持つデザイナー。現在はSega Pinball Incのデザイナーとして活躍中。今号の特集で紹介しているGolden Cueは彼の作品である。

このゲームの問題点

各年代の7作品を収録しており，値段も手頃(筆者は約5,000円で購入)。これでソフトウェアの完成度が高くバグもなければ素晴らしい作品なのだが，残念ながら完成度は異様なまでに低く，バグも複数存在する。

●ソフトウェアの問題点

パッケージの裏側に「精巧な物理モデルがボールアクションをコントロールし，実際のピンボールの動作をすべて再現しています」と記述されているが，これはまったくのでたらめ。本ソフトウェアで使用できないフリッパーテクニックが多数存在する。さらに致命的な問題として「ボールがフリッパーをすり抜ける」ため，動作のリアルさはまったくない。ナッジ(揺らし)も微妙な感覚で揺らすことはできず，微妙なボール捌きはまったく不可能。ピンボールの本質的な楽しみは「ボールを捌くこと」と「得点を組み立てる作戦性」だが，本ソフトウェアでは前者の点についてはまったく楽しめない。

Gottlieb社の作品は'90年代の作品もボールの軌道を決定しやすいランプやループには重点を置かない作品が多く，プレイヤーに微妙な球捌きを要求するゲームが多いので，この欠点は致命的だ。Spirit of '76に至ってはゲームにならない！　といい切れるほどだ。

フリッパーも狙いにくく「得点を組み立てる作戦性」以前の部分も問題が大きい。特にCue Ball Wizardでは致命的。実機では微妙に狙うときにはフリッパーに書かれている“Gottlieb”の文字をかなり参考にできるのだが，本ソフトウェアではまったくあてにもならない。フリッパーはフィールド内にある仕掛けの中で唯一プレイヤーの意志で動かせる重要な仕掛けであるのに，それを考えて開発したのか疑わしいできだ。

プレイ時とは別に，Baffle Ball，Humpty Dumpty，Knock Outではゲームオーバー時にボールがアウトホールに引っ掛かると，揺らしてもボールが落ちず，一度ゲームを抜けてメニューに戻

ボールがいっぱい入った状態。ボールは密着しているのにボール同士がヒットした効果音が鳴り続ける

る必要がある。筆者のパソコンは24倍速のCD-ROMドライブを搭載している環境だが，メニューに戻り再びゲームに戻るのに1分近くかかる。この現象は特に本ソフトウェアでは珍しいことではないのでしっかりデバッグしてほしかったところだ。

●マニュアル

各ゲームのルールについての説明が一切ないのはひどい！　付属してくる紙類もゲーム自体に関係のあるものはほとんどなし。ソフトウェアが世界共通のマスターCDのようで，まったく日本語化されてないのでこれは致命的。「スーパーのチラシ以下のマニュアル」といわれてもしょうがないできだ。パッケージに「ちょっとした息抜きにピッタリ」と記されているが，「ちょっとした息抜き」でルールが理解でき，遊べる程度のマニュアルを作成してほしいものだ。

パッケージには〝主な機能〞のひとつとして「ピンボールゲームのヒストリーをオンラインヘルプに収録(英語)」とあるが，これも非常に短いもので〝ヒストリー〞と呼べるようなものではない。さらに，本ソフトウェアと関連性のない部分ばかりで，存在する意義を感じさせない。それよりも各作品のフライヤー(いわゆるチラシ)など資料的なものや各時代の背景について(もちろん日本語で)記してほしかった。

手玉が左上まで行く。この作品の実機は国内でもプレイ可能なので，ぜひプレイすることをおすすめする。もちろん実機では揺らしにより手玉をここまで引き上げるのは不可能

こうなると揺らしてもボールは落ちず，再ゲームするには一度メニューに戻る必要がある

最後に

まだまだ不満がいっぱいある製品なのだが，今回は「マイクロソフトピンボールアーケードクレーム特集」ではないので，この辺で締めに入ろう。

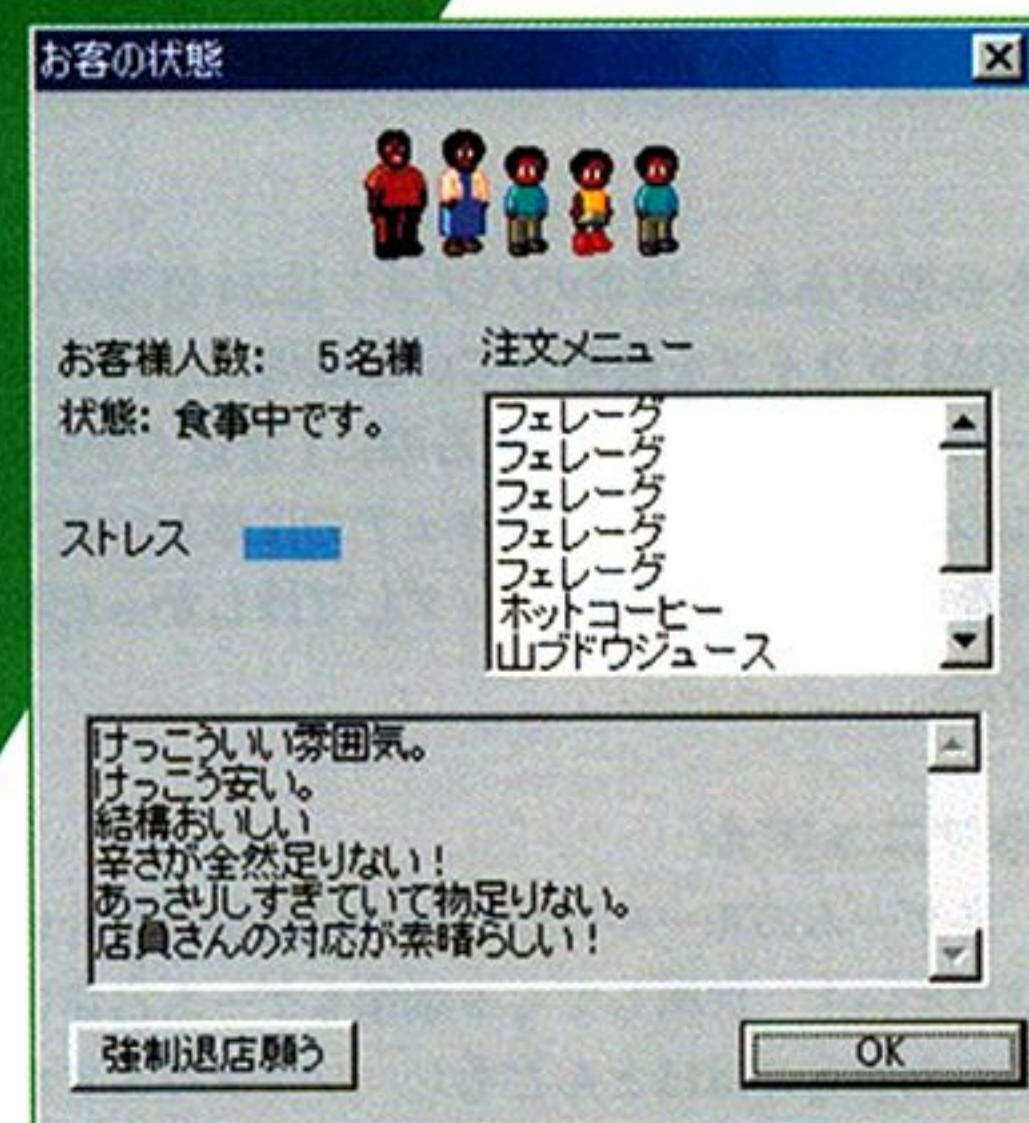
ピンボール以外のジャンルでもそうだが，一度移植された作品を同じプラットフォーム上で移植することは商品価値が極端に落ちるため大変難しく，ほとんど行われていない。移植を心がけるメーカーはこのことを踏まえたうえで，開発，販売してほしかった。

特に今回は発売元のマイクロソフトの唯一の長所である資金力を活かし，優秀な人材を投入してほしかったところだ。Baffle BallやHumpty Dumptyのような歴史的な作品を移植する場合には責任感を持って作業してほしいものだ，と同社の姿勢に改めて幻滅する。

非常に残念なデキだが，〝ピンボールの歴史やゲームの歴史に興味のある人〞，〝昔のゲームの好きな人〞，〝お金の余っている人〞は購入するのも悪くないと思う。

筆者も多くの不満を感じているのだが，すでにプレイ不可能な作品に，ごくわずかながらにでも触れる機会を得た気分になれるこのソフトウェアは，やはり購入してよかったと思っている。

ビストロ2に見る「食べ物ゲーム」の至福

西尾ゆき Nishio Yuki

人間の根源的な欲求である「食欲」。誰でもおいしいものを食べると幸せになれる。シミュレーションでもアクションでもとにかく食べ物を題材にしたものを「食べ物ゲーム」と呼ぼう。数少ない食べ物ゲームの中でも，好き嫌いの分かれる「ビストロ2」。だがハマる人はとことんハマってしまう。その欠点と魅力を通じて，ビストロシリーズの明日を探っていこう。

私は食べることが好きなので，ゲームでもそんな食べ物関連のものがないかと探すことが多い。しかし，星の数ほどゲームが散乱しているこのご時世なのに，料理を題材にしたゲームは意外と少ない。なかでも，調理に重点を置いているPCゲームはまったくといっていいほど見当たらない。ほとんどがレストラン経営ゲームである。たまに，お，食べ物ゲームか?! と色めきたってもレシピ集やワイン大全などのエデュテインメント系ソフトであって，がっくりさせられる。

見渡した限りで現在入手可能な食べ物PCゲームは「Pizza Tycoon(MICRO PROSE)」「The ビストロ(ピー・エス・ディー)」「The ビストロ2(ピー・エス・ディー)」「ファミレス(マスターピース)」の4つのみ。それぞれにいいところも悪いところもある。なかでも，ビストロシリーズは欠点が多いわりに人を惹きつける魅力にあふれており，最新作のビストロ2もシリーズのファンを夢中にさせてくれる。

強烈にダメなところが多いゲームなのに，なぜこれほどまでにハマるのか。内容を紹介するとともに，このゲームの長所と短所を徹底的に研究してみよう。

ビストロ2の主な内容

ビストロ2はレストラン経営のシミュレーションゲームだ。世界各国でレストランを経営し，たくさんの経験を積んで最終決戦の場，東京のレストランで五つ星評価を得ることが目標。プレイヤーはフランス，中国，インド，ケニア，アメリカ，イタリアのなかから国を選んで，レストランの経営を始める。

まずは内装やテーブルレイアウトを決め，店員を雇ったらメニューを選ぶ。ご当地料理から好きなものを好きなだけ選んでメニュー登録していく。その際，各料理を担当する調理師や味の濃い薄い，料理のグレード(仕入れ値が上下する)，調理時間なども決定する。これらは店を経営している間も，客のニーズにあわせて調整していく項目だ。

いよいよ店をオープン。初めのうちは，店員たちの足が遅く，接客態度のレベルや料理技術も低い。経験を積むうちに能力も高まっていくが，最初はお客が怒って帰るほどとろい。だが，やがて接客をこなすうちに魅力や機動力がアップしていくのだ。

なかなかお客はこないので，レストランを開いたばかりのときは食材を無駄にすることも多いのだが，店員が慣れてきて，広告もバンバン打てるようになると，何十人ものお客が押し寄せるようになる。

そして，来客数や売り上げなどの諸条件をクリアすると，晴れて星つきのレストランに。店のレベルが上がると椅子などの調度品，調理場やレジといった内部設備，壁紙や入り口などの内装/外装で選べるアイテムが増える。これらのアイテムを使って改装を重ね，お客の好みを研究しつつ，店のレベルアップを追求していくのだ。満足のいくお店ができたら，今度は違う国に移転する。そこで，また一から新しいレストランと従業員を育てていく。

このようにして世界各地の国でレシピやアイテム，究極の料理人を集めて東京で開店。いままで手に入れた経験やアイテムをすべて投入して世界一のレストランを目指すというのがビストロ2なのである。

ビストロ2の長所と短所

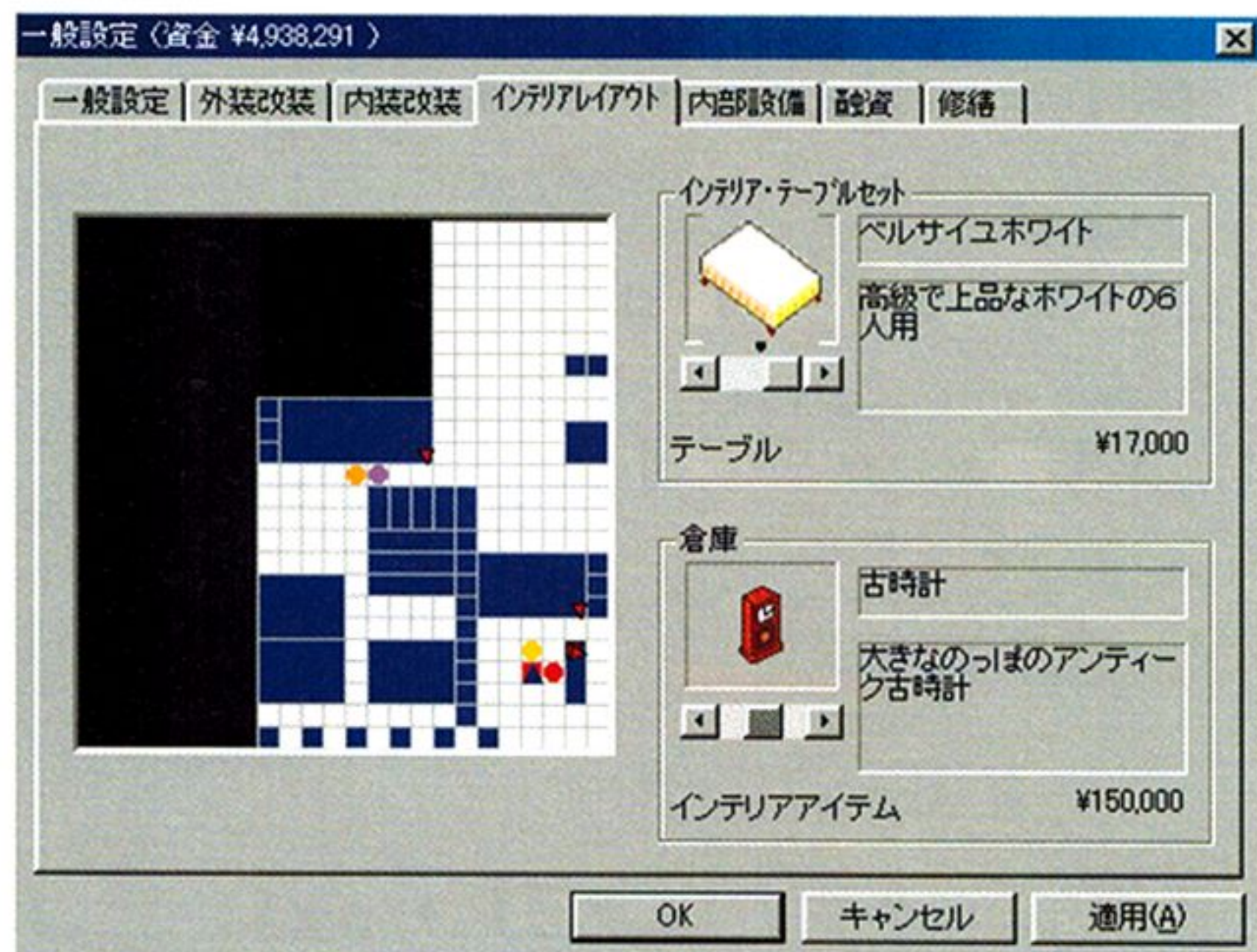
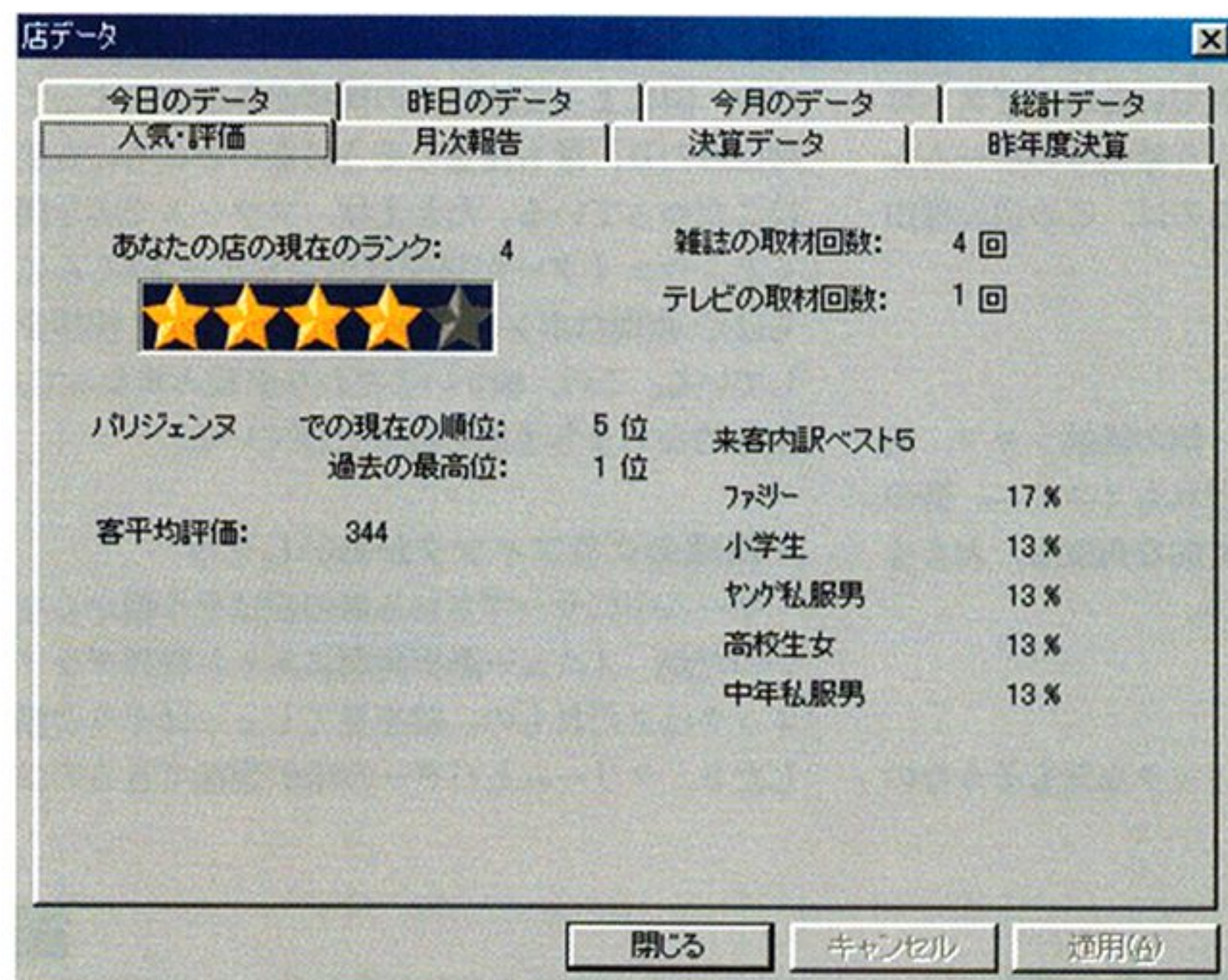
●欠点

あまりにも多いビストロ2の欠点。プレイヤーをもだえ苦しませる数々の出来事は，マゾヒステ

味にやかましいお客たち。おいしいまずい，濃い脂っぽいなどたくさんの感想を述べるので，徐々に店の味を調整していく

本当はおしゃれな店内にしたくても，客の好みや効率によって学食のようなレストランになりがち。さみしい

おいしそうな料理がズラリ。東京で店を開くときには各国で得た料理すべてがメニューに載せられる

最終決戦の場，東京ではいままで得たアイテムをすべて投入することができる。いろいろまぜるとカラオケ居酒屋に……

テーブルや植木の配置はちょっとしたドールハウス気分。レベルが上がるとピアノや彫像など，さまざまなアイテムが利用できるようになる

や，やっと四つ星獲得！　ここまでくるのに，何日かかったことか。なぜこんなにレベルアップが難しいんだ，このゲーム

東京では洋食メニューが手に入る。前作から流用されたグラフィック，ファンとしては懐かしいと喜ぶべきか，「まんまじゃん」と悲しむべきか

ィックなシミュレーション好きにはかえってたまらないのかも。ここでは数ある短所のなかから，その代表的なものを紹介しよう。これらが直れば一般にもアピールできる強力なシミュレーションゲームに変身するのか。それとも，絶妙なバランスがくずれて単なるつまらないレストランゲームに成り下がるのか。謎だ。

・よりシンプルに

広告など，プレイヤーが1週間やりたがるものを，5日，3日とわざわざ区切らせるのはおかしい。ビストロ内時間で毎日店を見ていてもらいたいのかもしれないが，ただでさえ手探りのゲームで，手間がかかるのは辛い。シミュレーション部分を充実させたいのならば，こういった省ける手間を入れるのはよしてほしい。

・バグ

ビストロの1にもあったバグが直っていない。また，仕様なのかどうかもわからないが，こちらの指定していない日に勝手に店が休みになってしまう。定休日を設けているにもかかわらず，レベルが上がるかどうかの瀬戸際にこれがあると最悪。店が開店せぬまま目の前で時間がささーっと進むのは，精神的ダメージが大きい。

・煩雑なインタフェイス

とにかくウェイターのテーブル担当の割り当てがしにくい。この辺のエリアはA君にと思っても，テーブルを1個ずつクリックし，さらにいちいち手間をかけないと担当を指定できない。ががっとドラッグかなにかで範囲指定したいものだ。メニューの登録も同様で，メニュー登録画面から品を選んで，もうひとつのメニュー設定画面で細かな設定を行う。もっとマウスの右ボタンを使うなどして簡略化してほしい。

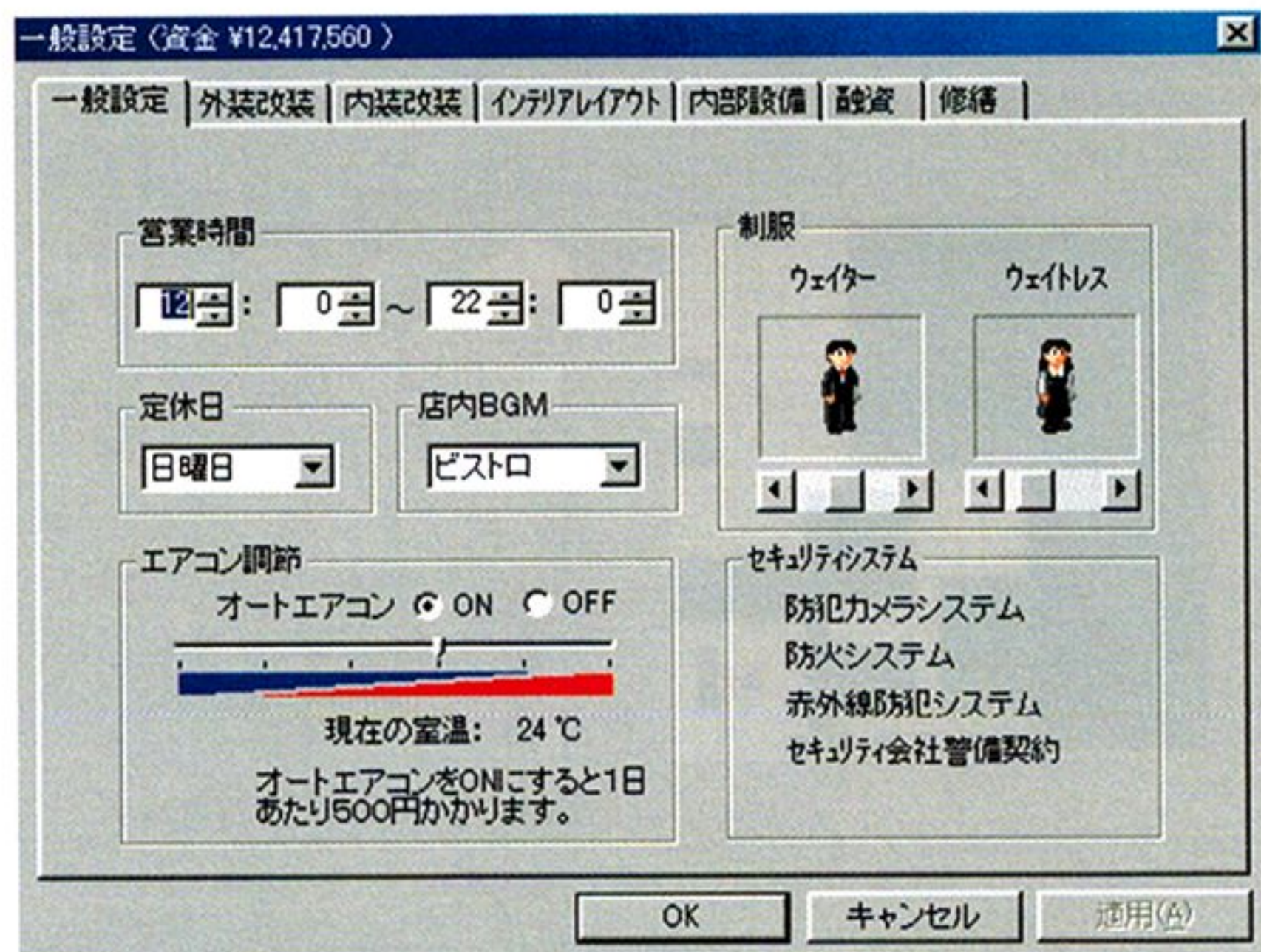
・戦略が立てづらい

隠しパラメータが多いため，なにをどうすれば客が増えるのかが非常にわかりづらい。客によって料理の感想や内装に対する感じ方もばらばらなので，いまひとつ「これだ！」というものがつかめないのだ。また，どの広告がどれだけの影響を持っているのか，さっぱり伝わってこない。探る楽しみがあるのは事実だが，探っても答えが出ないのでは欲求不満になる。ある程度判断材料が揃っていて，なおかつ攻略しがいのある難易度を持つというのがシミュレーションの理想だ。

・料理不足

仕入れた料理がきっちりはけると店は閉まるのだが，たとえ2品残っていても店は閉まってくれない。そんな半端な数が残っていると，決まって3人とか4人とかの客がきて「料理が足りないので怒って帰りました」などと客評価の数字が下がってしまう。もう少し，この辺は臨機応変に対応してもらいたい。

・ウィンドウの大きさが変えられない

ゲーム画面の大きさが変えられないため，15イ

給仕の制服が変えられるのも○。女性用にはなぜかバニーガール姿もあって，使用すると急にキャバクラ状態に

中国もなかなか雰囲気があっていい感じ。もちろんメニューにはおいしそうな中華料理がアレコレ並んでいる

ンチモニター，800×600ドットなどでプレイしていると，画面をビストロ2が占領してしまう。

・ほかのソフトと一緒にプレイしづらい

従業員が成長したり，食材が残り少なくなると情報ウィンドウがポップアップするのだが，このときいちいちビストロ2がアクティブウィンドウになって，ほかのソフトを使っていても画面が切り替わってしまう。箱庭ゲームの常として，ほかのことをしながらの「ながらプレイ」が求められるのに，これではパソコンに向かっている間はビストロ2しかできなくなってしまう。画面の端っこにあるビストロ2を見て「ふむ，今日は客の入りがいいねぇ」などとつぶやきながら，他のエディタや表計算ソフトなどをメインに使用していたいのだ。長く遊ぶためにも，この欠点だけは直してもらいたい。

●長所

理不尽な障害の多いなかでもゲームを続けてしまうのは，ひとえに以下に挙げるような長所が，食べ物好きな箱庭ゲーマーの心をわしづかみするからである。山ほどある短所にまぎれて批判だけされることが多いビストロ2だが，ミニチュア度は高く，コンセプトはしっかりしているので，かなり見所のある箱庭ゲームといっていいだろう。

・簡略化のしかたがいい

たとえば，従業員の休息室はどうしようとか，出勤管理はどうすればいいんだとか，帳簿づけが大変大変とか，実際のレストラン経営にあるであろう細かな仕事や難題はばっさり切り捨てられている。料理の仕入れにしても，食材別に仕入れるのではなく各料理の個数×原価分のお金を資金から引いていくだけのお気楽さだ。客を喜ばし料理を提供することに集中しながらも，レストランオーナーの気分は味わえるようになっている。

皿洗いや掃除までリアルタイムに追究しようとしたマスターピースの「ファミレス」と違い，レストラン経営という事柄からエンターテイメントを，またゲーム性を切り取れているのがビストロの素晴らしいところだ。かなり難ありのゲームながらも熱狂的なファンがいるのは，この辺に理由があるのだろう。

・ミニチュア度が高い

親父のマフラーからパリっ娘の縞縞シャツ，道端に駐車してある車まで，どれもミニミニ。街の一角を切り取ったような箱庭的な角度も，おままごとチックで非常にかわいい。

・変なところに細かい

各国の特色を出すグラフィックなどもそうなのだが，国によって給仕人の挨拶が各国語によって叫ばれたり，誰も注意しそうにないところにも変にこだわっている。たとえば，フランスで店を開くと，ウェイターが昼間はボンジュール（こんにちは），夜間はボンソワール（こんばんは）と挨拶をしている。この，細かいこだわりが積み重なって，全体的なちまちま感を盛り上げている。

・料理のグラフィックがおいしそう

ゲーム中にサーブされる皿の絵はそう細かくないのだが，メニュー選択画面にあるお料理グラフィックはよだれもの。絵を見てしょっぱそうと感じたり，クリームとバターの味が想像できるのは

前作　ビストロ

ビストロ2の前作「ビストロ」は，東京のさまざまな繁華街で店を作り，五つ星レストランを目指すレストラン経営シミュレーションであった。大筋において2と同じなのだが，毎日手動でエアコンの温度を変えなくてはならなかったり，客がハンバーグばっかり注文するから専門店になりそうだったり，客の入りが天気に左右されてそうでされてなさそうでさっぱりわからなかったり，とこれまた欠点が多い。しかし，こちらもやはり魅力爆発。理不尽なレベルクリアを果たしたあとの爽快感は，障害が大きいだけに2の上をいくかもしれない。

とにかくミニミニ感と色味がいかす。テーブルなどのリネン類の色も非常に心温まる。いかにも現実にいそうな服装の通行人や，ちょっと変なキャラクターを2頭身にせず描いているのもいい。メニューの料理画像もリアル系だが，味が濃そうでおいしそうだ。

前作もちまちま感が秀逸。ルーズソックスの日焼け女子高生や紙袋を持った怪しい秋葉原人間などが肉眼で確認できる

メニューはいかにも洋食屋さんといった感じのものが並ぶ。こちらは豚の生姜焼き。見ているとお腹が減ってしまう……

刺激的な言葉がほしい

食べ物ゲームはなにも実在のメニューにこだわる必要はない。大事なのは，いかにプレイヤーにイマジネーションを起こさせるかである。

そういう意味で心に響いたのが「プリンセスメーカー2」。娘を育てながらいろいろなアルバイトや稽古事をさせるゲームなのだが，年に1回お祭りがあって料理コンテストも開催される。これは特別なイベントではなく，単に娘の料理パラメータで判断されるだけのものなのだが，このコンテストに出品されるお料理の名前が刺激的だ。「毛蟹のグラタン小悪魔風」「地鶏と季節の野菜の鍋焼きヌードル」などなど。小悪魔風といわれるとニンニクかなぁとか，唐辛子かなぁとか考えてしまうし，クリーミィとかいわれると，ホワイトシチューやクラムチャウダーとか思い浮かべる。食べ物を表現する言葉は，使い古されたものでは新味がないし，あまり突飛なものでも共感できない。この辺のさじ加減は難しそうだ。

作っている光景もなかなかおいしそうだった，プリンセスメーカー2のコンテスト場面。娘が1位だと父親は小躍り

Pizza Tycoon

ここ4～5年に発売されたうちでイチ押しは，DOS時代の海外ゲーム，MICROPROSE社の「Pizza Tycoon」だ。これはピザ屋を経営してヨーロッパ中に支店を出し，各地をピザで征服してしおうという経営シミュレーションゲームである。現在も同社の復刻版シリーズの並行輸入品で入手可能。ぜひ一度遊んでみてほしい。

経営シミュレーションとしては経営部分がわかりづらく，当時はそれほど高い評価は得られなかった。だが，おまけ的要素の強かったピザ新製品開発モードは素晴らしいのひと言。

プレーンなピザ生地の上に，好きな材料をドンドン載せていくのだ。トマトや玉ねぎを載せ，ペパロニサラミを並べて，刻んだブラックオリーブを散らす。チーズも種類が選べたりして，もう最高。好みによってはロブスターをどどんと真ん中に置けてしまったり。できあがったものはその場で審査員に評価してもらえた。

ピザ作成画面。材料は左下のクッキングカッターで細かく刻むことができる。カロリーや脂肪分も表示される

コンシューマゲーム界の食べ物ゲーム

いままで見たなかではPSの「バーガー・バーガー」がピカ一だ。全体的にポップな色彩でまとめられた良品。ハンバーガーチェーン店の経営シミュレーションなのだが，新メニューの開発や会社組織・社員のレベルアップ，グレードが上がることによる特典など，エンターテイメント要素をしっかりつかんでいる。バーガーの具材の調理方法も揚げる・ゆでる・炒めるなど，多岐にわたっていて大興奮。キャビアや酢豚などハンバーガーにあるまじき材料も楽しい。

だが，特定の社員によってしかもたらされない具材が多いわりに雇用可能者数が少なく，社員育成と具材収集欲が反目するのは×。また，チェーン店を増やすための経営シミュレーション部分も終盤になると同じことの繰り返しでつまらない。メニュー開発部分はプレイしながらよだれがたらせる域に達していたので，この辺は少し残念。

非常にうれしい。

今回は世界で店を開くということで，各国の料理が楽しめる。イタリア料理やフランス料理などの比較的なじみの深いものから，アフリカの聞いたこともないような料理まで，さまざまなメニューに触れられるのも楽しい。

求む，ビストロシリーズ続編

さて，長所と短所を挙げ終わったところで，ビストロシリーズのこれからを考えてみよう。

まず，シリーズの続編が出るとすれば，いままで挙げた欠点を直したうえでさらなる改革が求められる。1から2へは細かい改良点，世界各国への展開などさまざまな変更点が認められるものの，基本システムはほとんど変わっていない。エアコンが手動から自動に変わったというのも前作からのファンならば小躍りするほど嬉しいが，2からシリーズに接した人間にはやはり単なる余計な手間だ。

また，ゲーム本体だけでなく，マニュアルやチュートリアルの改良も求められる。ビストロの開発元であるピー・エス・ディーのホームページは昨年から更新されていない。それ以前は開発者の声が読めたり，テクニック集，初心者向けのFAQなどが載っていたりと，いい雰囲気だったのに残念。現在は販売元のメディアカイトのサイトにちょっとしたFAQはあるが，充実しているとはいいがたい。もう少し，プレイヤーをフォローするような体制を整えてもらいたい。

いろいろ文句は出るのだが，それはビストロがこれからもっとよくなる可能性を秘めているからこそ。短所に挙げた例がなくなり，よい部分が増えていけば，きっとゲームの歴史に残る食べ物ゲームになれる。

よく描かれたグラフィック，癖になる音楽，そして料理を人に提供する楽しさを味わわせてくれるシステム。ビストロ2にはほうっておけない，キラリと光る部分が多いのだ。

これだけ食べ物ゲーム砂漠なPCゲーム業界でビストロシリーズは私たちの最後の希望の星だ。ちょっとずつでもいいから，改良してもらって，末永くこのシリーズを続けてもらいたいというのがファンの望みなのである。

The ビストロ2
開発・発売元：ピー・エス・ディー
販売元：株式会社メディアカイト
問い合わせ先：☎03-5449-3181
support@media-kite.co.jp
OS：Windows95/98
CPU：Pentium/133MHz以上
メモリ：32MB以上
ビデオ：640×480ドット以上の解像度，256色以上のカラー表示
HDD：20MB以上
その他：DirectX ver.5.0以上

究極の箱庭か?ポピュラス・ザ・ビギニング

西川善司 Nishikawa Zenji

発動させるスペルの射程距離は同心円で表示される

**1998年12月18日, 待望の「ポピュラス3」こと「ポピュラス・ザ・ビギニング」(以下ポピュラス3と表記)の日本語版が発売された。
シームレスでオブジェクト指向な斬新なユーザーインタフェイス, 妥協なしのパワーグラフィック, そして練り込まれたゲームデザイン……と, 究極の箱庭&ストラテジーゲームとして早くも全世界のゲーマーを虜にしている。
このゲームの魅力を探ってみることにしよう。**

ポピュラス・ザ・ビギニングって?

ポピュラス3を知らない人もいるかもしれないので, まず基本ルールを紹介しておこう。

ゲームの目的は単純明快。ステージとなる惑星の上に住む自分の種族を繁栄させ, 敵種族を全滅させること。

プレイヤーはこの世界の「神」となり, その神が行えるのは, 住民に土地に建物を建てさせること, 住民を誘導すること, そして住民たちのリーダーとなる「シャーマン」を通してスペル(魔法)を発動させ, 天変地異を起こさせること, の3つだ。

スペル(≒天変地異)には敵種族や敵種族の土地や町を破壊する「攻撃タイプ」と, 自分の種族を守ったり, 自分の種族の土地や町の拡張を手助けする「防御タイプ」の2つがある。これらを自らの戦略に基づいて駆使し, 敵の「神」を崇拝する敵種族を打ち負かさなければならない。

スペルのパワーの根源となるのはプレイヤーつまり「神」に対する信仰心であり, これは自分の種族の人口に比例してくる(*1)。スペルを使うとマナ(魔力)を消費してしまうが, 人口が多ければマナの回復は早い。逆にいえば, 敵種族のマナを奪うためには敵種族を殺しまくればいいことになる。

それでは人口を増やすにはどうしたらよいのか。自分領土の住民を増やす方法は基本的には2つ。

ひとつは地上を走りまわって気ままに暮らしている「野人」を「コンバート(回心)」のスペルで自分側の種族にしてしまう方法。

もうひとつは建設した住居小屋に人民を住まわせて増殖させる方法だ。

このほか, 敵地に伝道師を送り込み, 接した敵種族を洗脳して味方にしたり, 混乱の魔法で一定時間, 味方種族にしてしまうようなアグレシッブな戦術もないこともないが, 確実な方法ではない。

どうやって敵よりも人口を増やすか……ここにポピュラスというゲームの戦略の出発点がある。

*1:POPULOUS(「人口の多い」)というゲーム名はここからきている。

ゲームの流れ

人口を増やすためには, ひたすら, 自分の種族の町の建設に挑まなければならない。町の規模が大きくなればなるほど人口も増えていく。

人口が適度に増えてきたら次にやるべきことは兵力の増強だ。そして「戦争」。

ゲームの勝利条件は敵種族の殲滅。ひとりも生存者を残してはいけない過酷な戦争をしなければならないのだ。

自分に仕える人々を訓練し, 特殊能力を身につけさせ, 軍隊を編成し, これを敵地に送り込むことになる。

訓練所にはいくつかの種類があり, それぞれの訓練所で, 格闘戦に強い「戦士」, 敵を遠距離砲撃できる「炎の戦士」, 敵を説き伏せて寝返らせてしまう「伝道師」, 敵の目を盗み敵地を焼き払う「スパイ」といった, それぞれ異なった特殊能力を持つ兵を作り出すことができる。

それぞれの兵には長所と短所があり, その長所を最大限に, その短所を最小限にするように行動させるのが, 彼らの大いなる潜在意識となる「神の意思」すなわち「プレイヤーの戦略」ということになるのだ。

そして巨大な人口が生み出す「神を敬う心」は強大なパワーを育む。これが天変地異の根元となるマナ(魔力)だ。この絶大なパワーを実際の奇跡や天変地異に変えるのが, 神の代弁者, プレイヤーの分身ともいうべき存在「シャーマン(巫女)」だ。

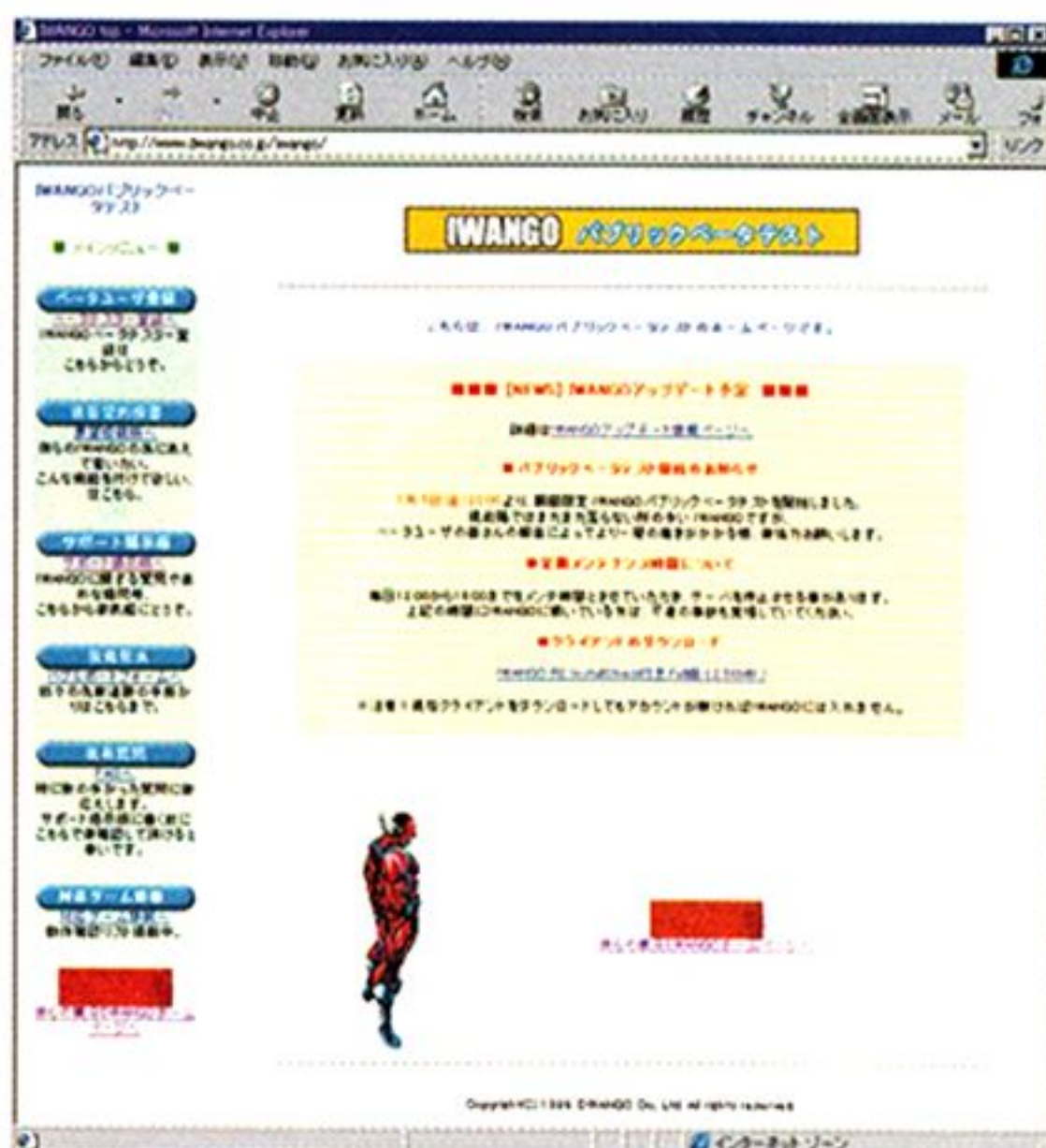
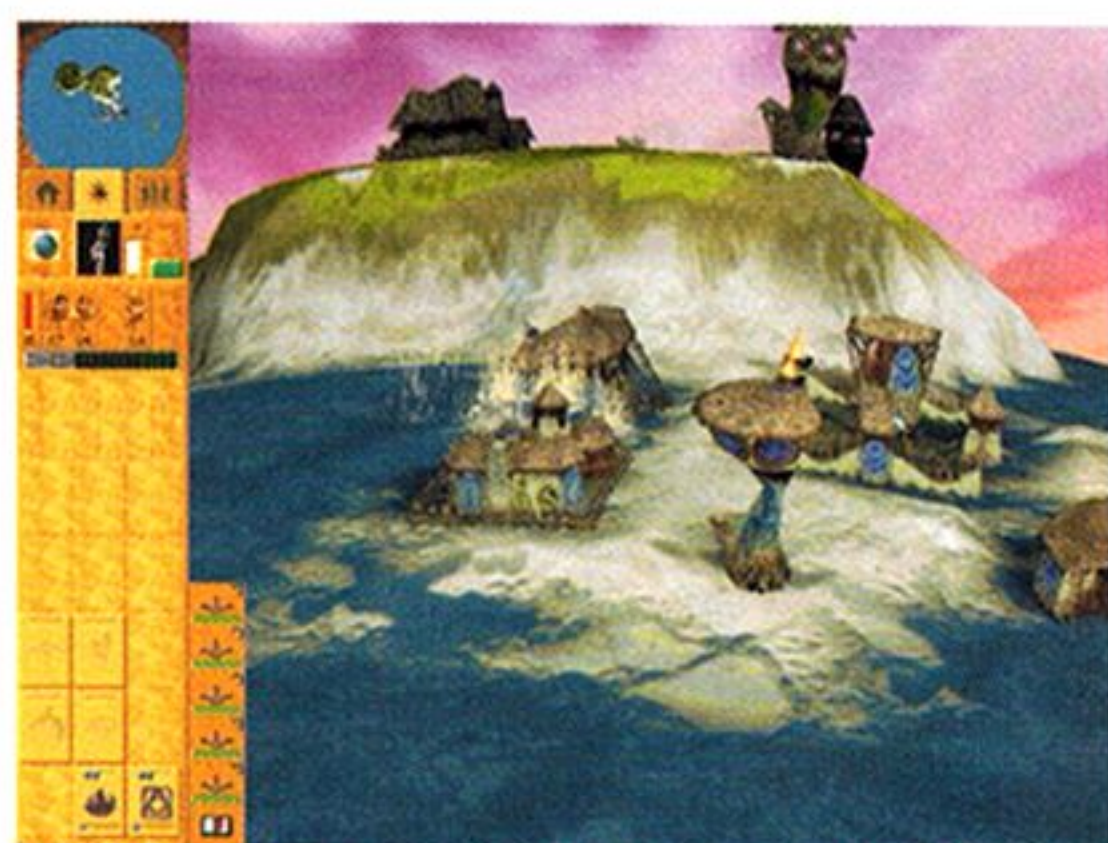
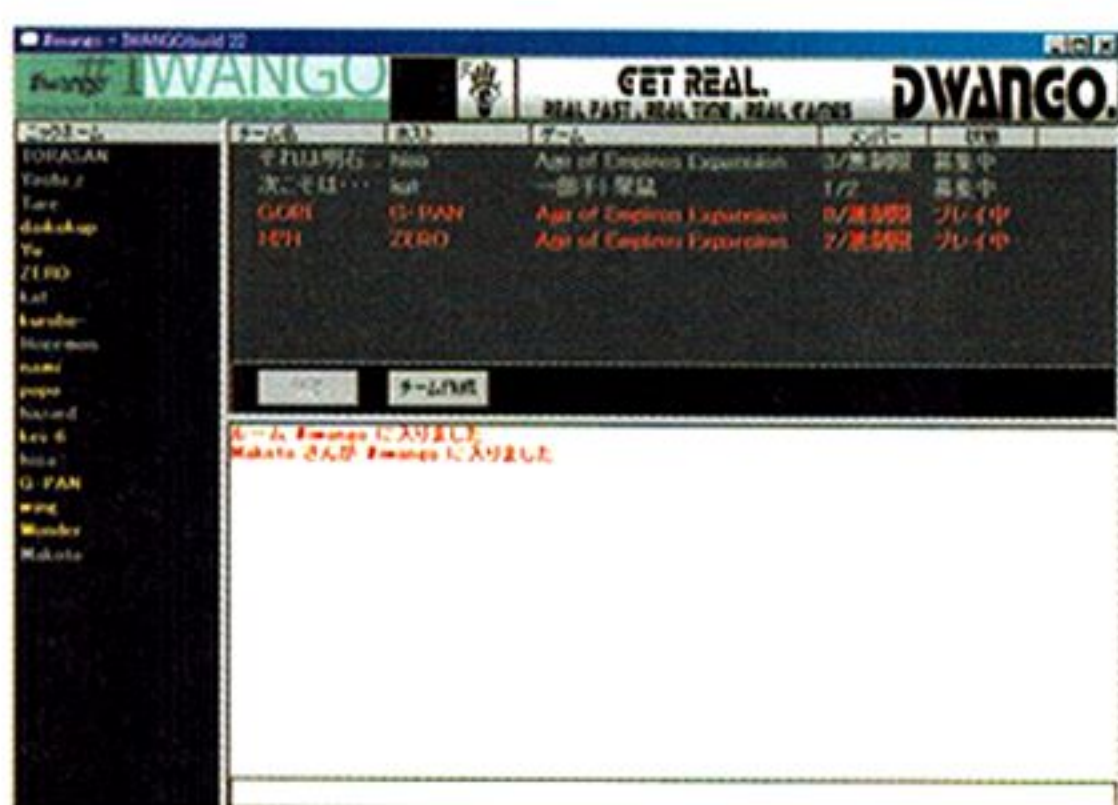
シャーマンは敵地に火山噴火や竜巻などの天災を呼び起こしたり, 地形そのものを変えてしまったり, あるいはその神の力で自分たちの軍隊を守ったりすることができる。神の力を具現化できるのは彼女だけの特権だ。

戦争を仕掛けて敵を全滅させるか, 天変地異で敵を全滅させるか, それともその両方か。この世界と自分に仕える住民たちの行く末は神であるプレイヤーにゆだねられているのである。

ポピュラス1&2とはどこが違う?

「ポピュラス1」(1990年発表), 「ポピュラス2」

こういった地面の造形が気軽に行えなくなってしまったのが「ポピュラス」シリーズ最大の変更点といえる

シャーマンというプレイヤーの分身的存在の導入がゲーム性に大きな変革を与えている

populous.netの画面

敵の執拗な「浸食」スペルの連続攻撃に，自軍陣地の建物が次々と水没させられている様子

IWANGOの画面

ここは新しいスペルの知識や建物の設計図を与えてくれる「知識の部屋」。当然，ここを巡っての激しい攻防が繰り広げられることになる

ボートで伝道師部隊を敵地へ上陸させたところ。敵の侵攻に備えて前線警備にあたっていた敵戦士たちは伝道師の諭しの声に聞き入ってしまう……

(1992年発表)は，Amiga，IBM PC，PC-9801，X68000，Macintoshといったパソコン，さらにスーパーファミコンやメガドライブ，PCエンジンなどのゲーム機にも移植され，ストラテジータイプのゲームとしては空前のヒットを記録した。

そういうわけで，ポピュラス1&2のプレイヤーは相当おられるはずで，そういった人たちにとっては「前作からどう変わったのか」という点に興味が集中することだろう。

まず，ポピュラスが「箱庭的ゲーム」といわれた根源である「土地の造形」が気軽に行えなくなってしまったこと……これが最大のゲーム性の変更点として挙げられる。1&2では土地の盛り下げはマナをそれほど消費せず行えた。もともと「海」となっている無の領域に地面を作り出し，そこに人を住まわせていく……というのが1&2の基本戦略だったわけだが，ポピュラス3では土地の盛り下げは結構な量のマナを消費してしまうため，むやみやたらに使えなくなってしまったのである。よって1で使えた，「自軍と敵陣を序盤から接続してしまい，敵陣を『土地の造形』で強引に盛り上げて破壊する」……というあの「凶悪テクニック」は使えない。

さらにポピュラス3ではスペルに「射程距離」の概念が導入されてしまった。これも大きい変更点だ。1&2ではマナがたまり次第，敵陣になんの前触れもなく火山を作ったり地震を起こしたりすることができたわけだが，3ではスペルの有効範囲が神の代理人であるシャーマンを中心とした同心円内に限られている。射程距離はスペルの種類によってだいぶ異なるが，いずれにせよ，敵陣のど真ん中に火山を打ち立てたりするには，シャーマンを敵陣まで導く必要があるというわけだ。

敵陣に近づけば敵に殺される可能性も高くなる。種族のリーダー，神の代理人シャーマンといえど，魔法が使える以外はちょっと耐久力が高いだけの普通の人。取り囲まれてリンチを受ければたちまち死亡する。そして自軍のシャーマンが殺されれば多大な魔力が相手側に充填される(*2)。つまり，マナさえたまれば攻撃し放題だった前作とは違い，攻撃する側にはリスクがつきまとい，攻撃される側にはマナ獲得のチャンスがやってくるというゲーム性に変更させられたということだ。1&2を知っている人間としては大胆なルール変更に思えるだろうが，ゲームバランスの設定としては3のほうがよりスリリングになっているといえる。

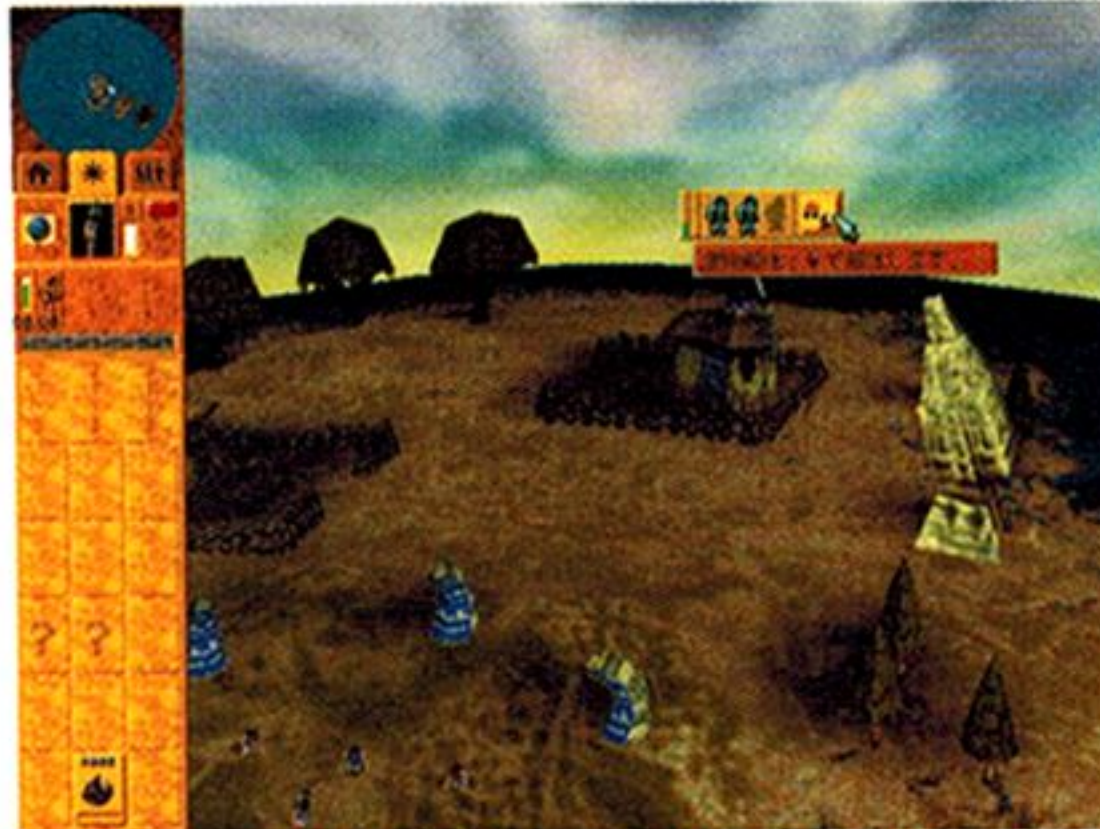
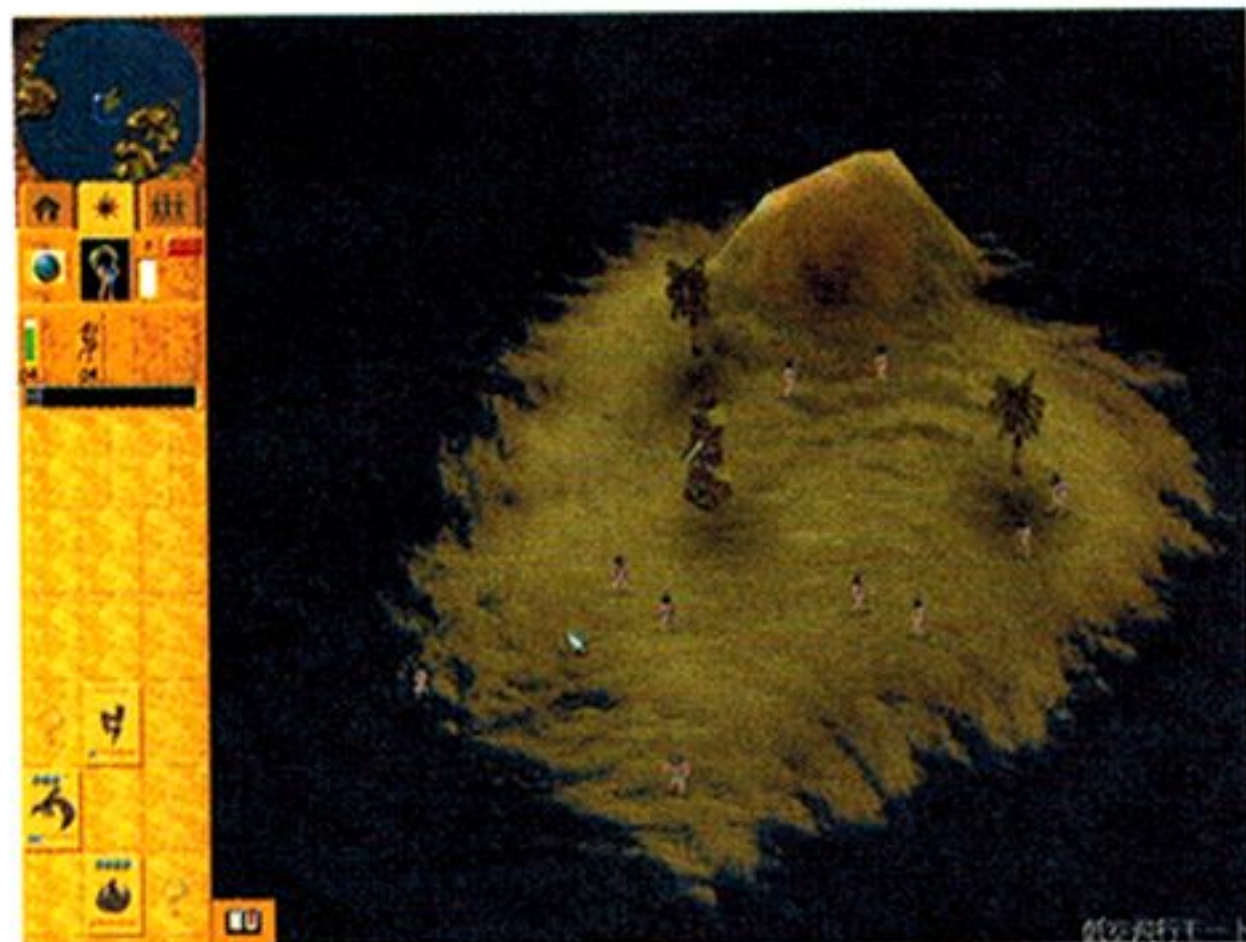
しかし，予測されるのが「神と神の戦いだったもともとのポピュラスの世界観とだいぶ異なるのでは?」という意見。

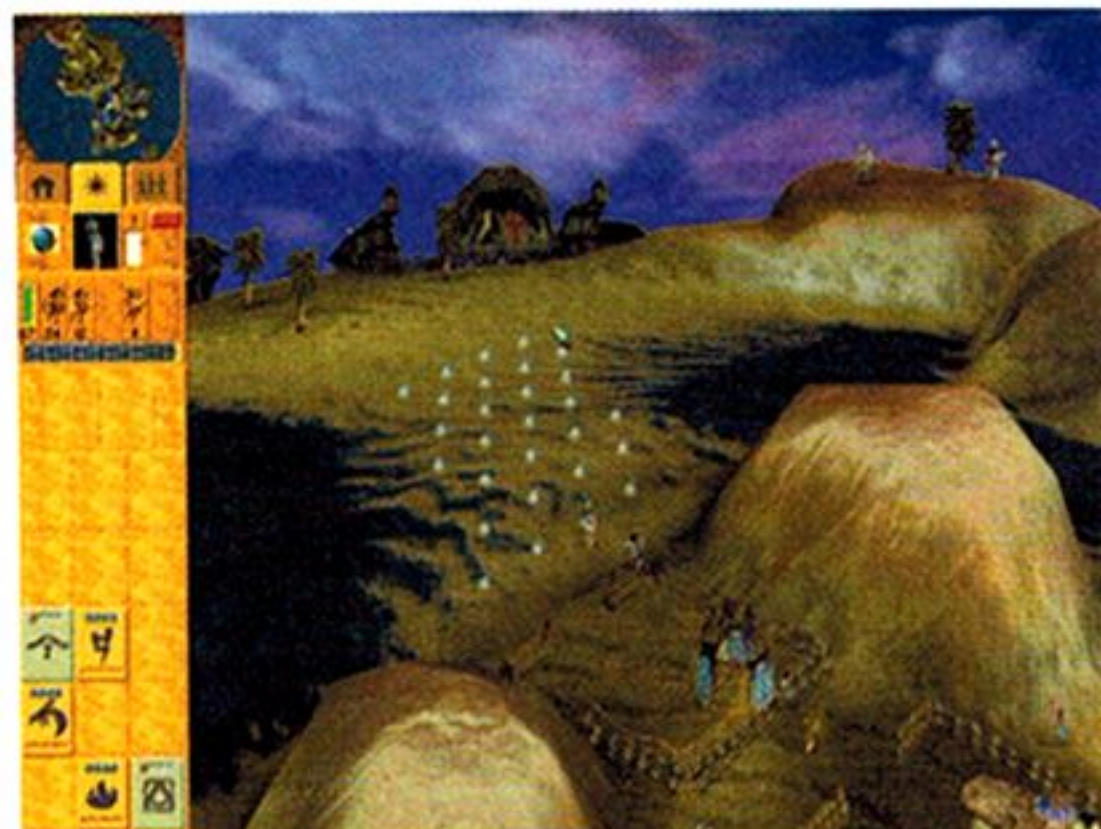
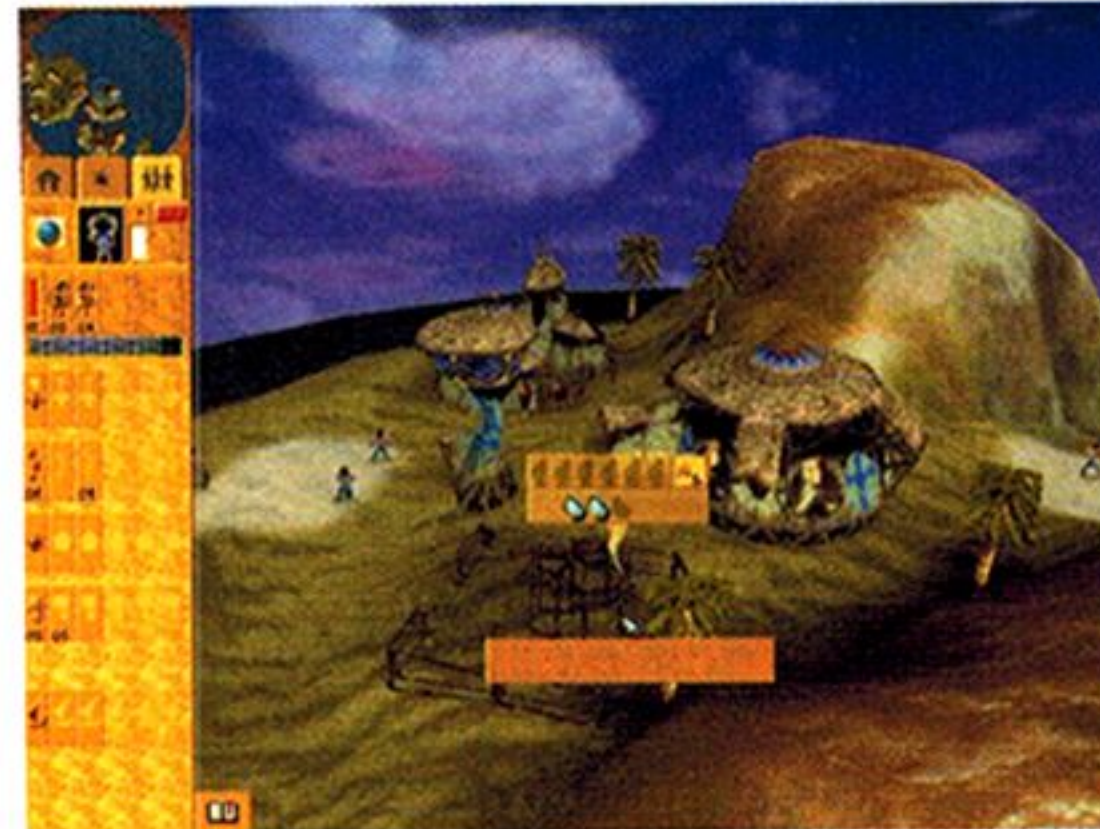
これはよりリアリティを追求した結果こうなった……と好意的に解釈できないだろうか。ポピュラス3では，神(＝プレイヤー)を雲の上の存在とし，人間界に直接手は出せない……という設定にしてしまったというわけだ。その代わり神との交信役シャーマンに魔力を与え，すべての住民に「敵の神を敬う敵種族を殲滅せよ」「そして繁栄せよ」と，その潜在意識として働きかけるというイメージだ。

＊2：シャーマンは死亡しても，味方住民が生き残っていればしばらくして味方本拠地で復活する。

ネットワーク対戦が熱い

ポピュラス1&2ではモデムやシリアルケーブル接続による対戦しか行えなかったが，ポピュラス3では，ついにインターネット上やLAN上での対戦ができるようになった。

シングルプレイヤー(1プレイヤー)モードでは，人間のプレイヤーがやや不利な状況に置かれて，

このような小屋に従者を住まわせておくと，新しい子が早く産まれて人口が増えていく。右はさまざまな奇跡を与えてくれる「ストーンヘッド」と呼ばれる石像だ

シャーマンが檻に捕らわれてしまい，これを救出しなければ攻撃スペルが使えなくなってしまうようなステージもある

「土地隆起」スペルで海に陸地を作り出したところ。敵地への侵攻路を探り出すのに最適なスペルだが，敵もここを使って攻めてくる可能性もある。諸刃の剣といえる

建設プランマーカー（左右に見える白いマス）を地面に配置していくことで建物を建設していく。建築には木材が必要で，木が肥沃な土地ほど建物が早く建てられる

強力な攻撃スペル「火山噴火」を使ったところ。マグマに触れた建物は焼けてしまい，地面は建設不能な焦土と化す

これを「どう打開していくか」という戦略パズル的なゲーム要素が強い。これに対してマルチプレイヤー（ネットワーク対戦）モードではすべてのプレイヤーが対等な条件でゲームが開始されるため，ゲーム序盤は水面下での戦略行動の応酬で非常に緊迫したムードが漂い，そしてゲーム後半は強力な攻撃スペルが飛び交う超過激な展開になる。

ポピュラス1のときもそうだったが，対人戦のときのポピュラス3は，1人プレイの何倍も面白く，そして熱いのだ。

さて，このネットワーク対戦だが，インターネットに接続する環境があればすぐに始められる。'99年3月現在，大手の対戦ポピュラス3のサイトは海外にひとつ，国内にひとつだ。

海外のほうはゲームの開発元Bullfrogが運営しているサーバーで，アドレス（URL）はズバリ，

http://www.populous.netだ。

ここにアクセスし，ユーザー登録をすれば，すぐに見知らぬ海外の人と対戦が行えるようになる。入会金，年会費などはなく，ゲームの製品版さえ持っていれば誰でも参加可能なので未成年者もOKだ。ただし，参加者は海外の人ばかりなのでサイト内でのチャットの公用語は英語になっている。

なお，日本語版ポピュラス3は海外版のバージョン1.01（バグフィクスパッチを当てたあとのバージョン）とのみ互換性があるので，対戦開始前には相手のバージョン番号を，

My Populous is version 1.01.
There is no compatibility between different versions.
Which version do you have?
（私のポピュラス3はバージョン1.01だよ。バージョンが違うと互換性がないんだよ。君のはバージョンいくつ?）

などといって確認したほうがいい。

一方，日本国内の対応サイトとしては対戦ゲームサイト「DWANGO」のインターネット対応版である「IWANGO」がある。URLは，

http://www.dwango.co.jp/iwango/

だ。ここで無料で提供されているIWANGO専用のクライアントソフトをダウンロードし，ユーザー登録を行ってから，そのクライアントソフト経由で対戦サイトと接続することになる。

ここは日本人ばかりなので日本語チャットがOKだ。IWANGOはポピュラス3以外のゲームにも多く対応しているため，アクセス者が必ずポピュラス3のプレイヤーというわけではない点に注意してほしい。

いずれのサイトにおいても，1対1はもちろん，最大，同時に4人までがプレイ可能で，3人以上のプレイヤーがいるときには1対2，2対2などの同盟を結んでのチーム対戦もできる。

ポピュラス3の対戦はひとりプレイモードとはまったく違った面白さがあるので，ぜひとも一度体験してほしい。

西川善司書き下ろし（?）の攻略本，4月下旬発売！

ポピュラス1の全盛期，ポピュラス1の作者であるピーター・モリニュー氏との日英エキシビションマッチで勝利してしまったことがある。

このことを覚えていたソフトバンクの編集者の計らいでポピュラス3の攻略本の執筆を依頼され，現在この作業を行っている。ソフトバンクから'99年4月下旬の発売予定だ。

本当はアメリカで発売された攻略本の和訳監修の任を受けたのだが，原書のデキがあまりよくなかったので90%を私自らが書き下ろしてしまっている。表紙には「監修」となっているかもしれないが，実質「著者」と思ってほしい。

内容は，シングルプレイヤー（1プレイヤー）用全25ステージの完全攻略のほか，マルチプレイヤー（ネットワーク対戦）用全15ステージの攻略，ネットワーク対戦の始め方，そのほかプレイ中に偶発的に発見した裏技の数々を満載したものになっている。かなり気合を入れて書いているので，ポピュラス3のプレイヤーにはぜひ読んでもらいたい。

また，最近入ってきたニュースによると6月にはPlayStation版の「ポピュラス3」も発売される予定らしい。内容に大きな差がなければ，こちらにも対応した版も起こす予定でいる。

最後になるが，とにかく「最近，なにか燃えられるゲームがないよねぇ」とお嘆きのゲーマー&昔ゲーマーだった諸君，とりあえず「ポピュラス3」を始めてみなさいって。　（西川善司）

Duel Fighter 2

今回紹介するのは，team無限うなぎの作成したX68000用の縦スクロールシューティングゲームだ。Duel Fighter 2という名前だが，1についてはよく知らない。FD4枚組の大作である。

キャラクターなどは3Dでレンダリングしたものを基本にして使っているようで，アニメーションパターンなども十分。もちろん(？)，DoGA CGAシステムが使われている。

ゲーム内容はご覧のとおりの横画面タイプ縦スクロールシューティングゲーム。敵の配置は2レイヤーになっている。攻撃はいわゆる地上空中別方式で，武器のパワーアップなどはない。自機はレーザーソードを装備しており，特殊攻撃として敵に斬りつけたり，弾を相殺したりできる。ちなみにダメージはシールド制でボムはない。

動作環境はやや高度で，X68000/16MHz以上を推奨，メモリ4MB以上。ハードディスクは必須。操作はジョイスティックのみとなる。twentyoneも必須だ。

操作はボタンそれぞれでバルカンとミサイル(それぞれのレイヤーを撃ち分ける)に同時押しでソード(弾消し，密着した敵に大ダメージ)となる。ちなみにHELPキーを押すと丁寧な操作系解説が出てくる。

ゲーム内容

ゲーム内容は画面からだいたい想像がつくようにレイフォースっぽい感じで，一部イメージファイトで，一部スターブレードっぽく展開していく。舞台は大気圏外の戦闘から大気圏突入，地上面，また大気圏外へ，宇宙艦隊面，ワープ，最終要塞面と展開し，最後はクリスタルコアを壊して終わり。展開としては一般的なところだろうか。

特徴のひとつにライバルキャラの存在が挙げられるだろう。可変型アーマロイドといった感じのキャラで，遠くからビーム砲をどかどか撃ったり，画面を端から端までビームでスキャンしてくれたり，何度も目の前に現れて弾をばらまいてくれる。

画面上に警報や各種メッセージなども表示され，結構ダイナミックにストーリー(？)は展開していく。

画面にばらまかれる弾数なども比較的多く，弾の速度は普通に避けられる範囲内である。基本的に避けるゲームだろう。

ただ，こちらを狙って撃たない敵も多いので，下手に大きく動かず，弾の間を縫うなり，弾消しをしていくことになる。

敵キャラクターは多彩で，形状は作り込んであるのにザコキャラでもったいないのとか，いろいろいる。ブロックの中ボス(？　でもないか)あたりは面白い。通せんぼをしているブロックがばらばらになって動き回り攻撃をしてくる。ブロックをよけつつ破壊していくことになる(赤いマークのブロック2個を壊せば全部消える)。各ステージの最後のボスキャラは少し大味な感じ。

あとは，ソードの使い方でゲームは変わってくる。

赤くてちょっと硬くて弾をバラバラ出す「普通のゲームだったらアイテムキャリアなのになあ」って奴とかが出てきたら，すっと近寄ってバサっと斬って捨てる。この辺を覚えてくると面白さがまた変わってくる。調子に乗ってると痛い目にあうのだが，ボスの弾の撃ち方などを読み切って弾消しを兼ねてザリザリ削る(間違うと即死か)。ボスキャラはかなり硬いのでまともにやってるとしんどい。シールドが残っていたらやってみるべきだろう(コンティニュー制限がないのであまり深い意味はないが，ステージの初めから繰り返しなので残機数次第か)。それとももっと緻密に避けまくって倒すべきだろうか(そうなんだろうなあ)。

後半になるとたいてい画面の半分しか撃ちきれないので，前方をクリアしたらあとはひたすら避ける(単に腕の問題かなあ)。

ゲーム構成の問題

デザインとして見ると，2レイヤー構成になっている意味がほとんどないように思われる。同じ輝度で描かれた絵だと，どのレイヤーの敵か判別できず，覚えていくしかない。下のレイヤーから拡大しつつ上がってくる敵などもいるのだが，下にいっぱなしのものもいて，さらに下にいるけど攻撃してこないものもいる。

撃ち分けるのは非常に辛い。

自動連射は非常によく出るので(出っぱなし)楽でいい。が，撃ち分けは非常に難しい。自動連射以上に効率よくボタンを押すのは不可能なので，事実上バルカンは押しっぱなしにせざるをえない。地上攻撃したいときだけミサイルを押せばいいかというと，それだと同時押しになるので切り替えが必要になる。すると弾幕が薄くなるので，撃ち分けを考える人はほとんどいないだろう。バルカンで撃ってみて効果がない敵はミサイルで攻撃する……などとやっているより，結局は直接の当たり判定(衝突判定)がある空中キャラだけ撃っ

こいつがライバルキャラだ

上のデカイキャラとかはソードでバサバサ斬りたいところ

てることになる。
　バルカン撃ちっぱなしで，緊急回避でソードを使う，というのがどう考えても一般的なやり方になる。下レイヤーの敵だけ出てくる場面は除いて，下からの攻撃は避けるだけに専念したほうが間違いがない。細かな撃ち分けを要求されるデザインでも鬱陶しいだろうし，2レイヤーにしたことはほとんど報われてないのではないかと思われる。
　また，ゲーム開始直後しばらくのあいだ出てくる敵はどれもミサイルでしか攻撃できない。その辺りは教育的なのか，混乱を招くだけなのかよくわからない。下レイヤーの敵はプレイヤーとは別レイヤーながら，空中キャラと同じように直に当たる弾を吐く。その辺りがレイフォースなどの立体デザインとはちょっと異なる。あからさまに別レイヤーとわかればいいのだが，見た感じ空中キャラだとほとんど区別できないものが多いのだ。
　基本的に2レイヤーのゲームとして作り始めたのだろうが，ソードの特徴に絞ったデザインにしたほうがまとまりが出たのではないかと思われる。

技術的な展開

　スプライトはかなりの数を出しているが，うちの中残光ディスプレイで見てもちらついている部分などがあり，一部かなり苦しい感じ。パートタイムで表示している部分はなくてもゲーム展開に支障はない。削るオプションがあってもよかった感じだ。
　エフェクト面ではいろいろ凝ったことをやっている部分も多い。
　背景は地上面の一部で弾の判定がほとんどできない色づかいなところを除けば，かなり良好。特に地上からまた飛翔していくシーンなどは非常に美しい。
　ワープ面はギャラガさながらの激しいダンスが繰り広げられる。綺麗だけど厳しい。敵のワープアウトしてくる位置がわかるとはいえ，大きく避け回りつつ倒していくしかない。ワープ空間で自機の航跡がビーム状に残るのも新鮮な表現かもしれない。
　そのほかアクティブ垂直帰線期間待ちモードとか，変な技術で，画面のちらつきかゲーム展開の円滑さかを優先できるようになっている。CPUパワーをかなり消費するゲームなので，非力なマシンでは必須なのだろう。
　FD4枚組でハードディスクへのインストールは必須という点はどうだろうか。エンディングアニメーションなどがあるのでこれはしかたないのかもしれないが，FDからそのまま遊べる形式のほうが手軽で望ましい。そのほか，BGMのバインド方法などが解説されている。「できればもっとほかの曲に変えてゲームしてね」というのが作者の意図のようだ。

　この作品は1997年制作ということで，最新のゲームというわけではないが，Oh!X休刊後にも同人ソフト界はもちろん健在で数多くの作品が作られている。まだまだいろいろなゲーム/ツールが作られているはずだ。もちろん，X68000のものに限らないので紹介希望のものを編集室まで寄せてほしい。よろしく。

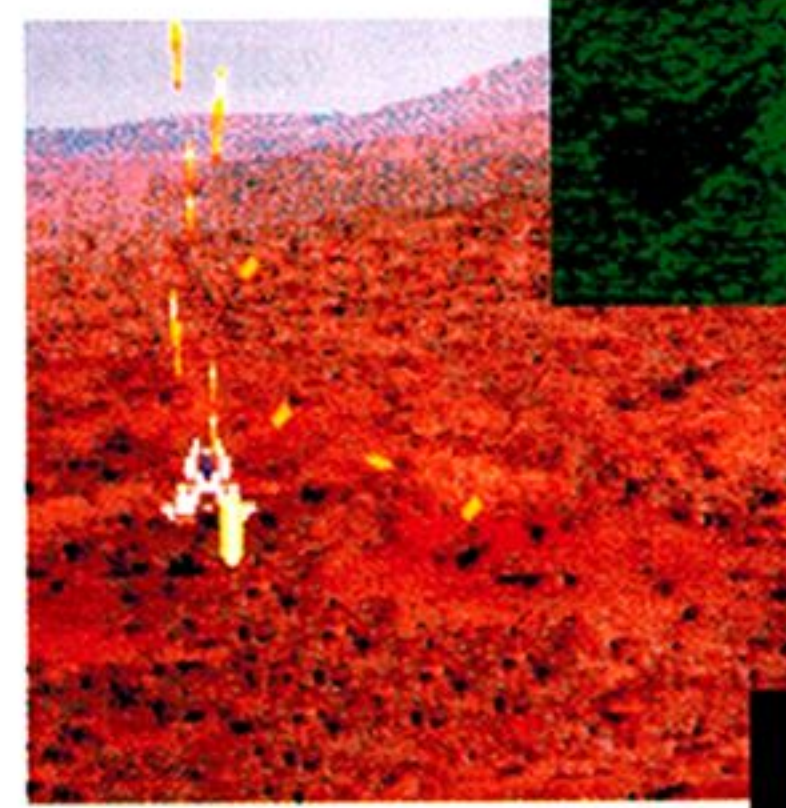
STAGE3のボスキャラ。弾をばらまくだけでちょっと大味か

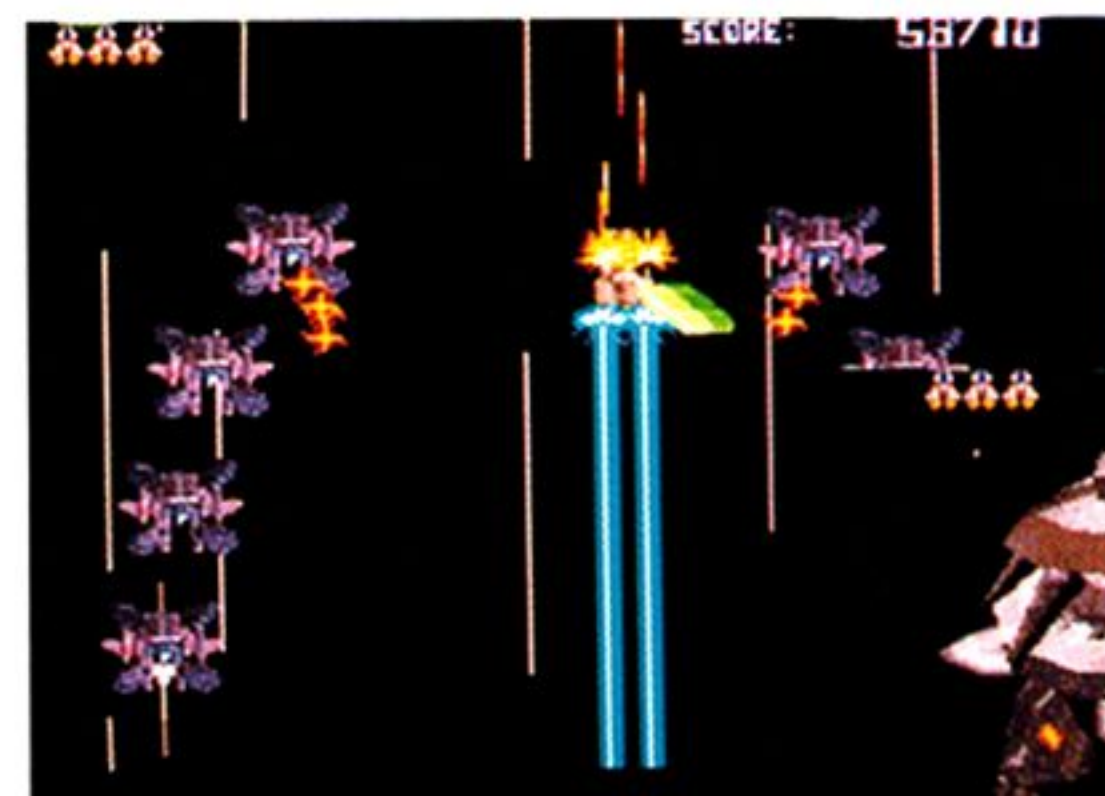
ライバルキャラのビーム攻撃。このまま端までスキャンする

地上へ降下中。レーザーは凄く痛いので要注意

この辺りは敵の弾がかなり見えにくい

ブロックが分離して襲ってくる。ソードで一気に片をつけたい

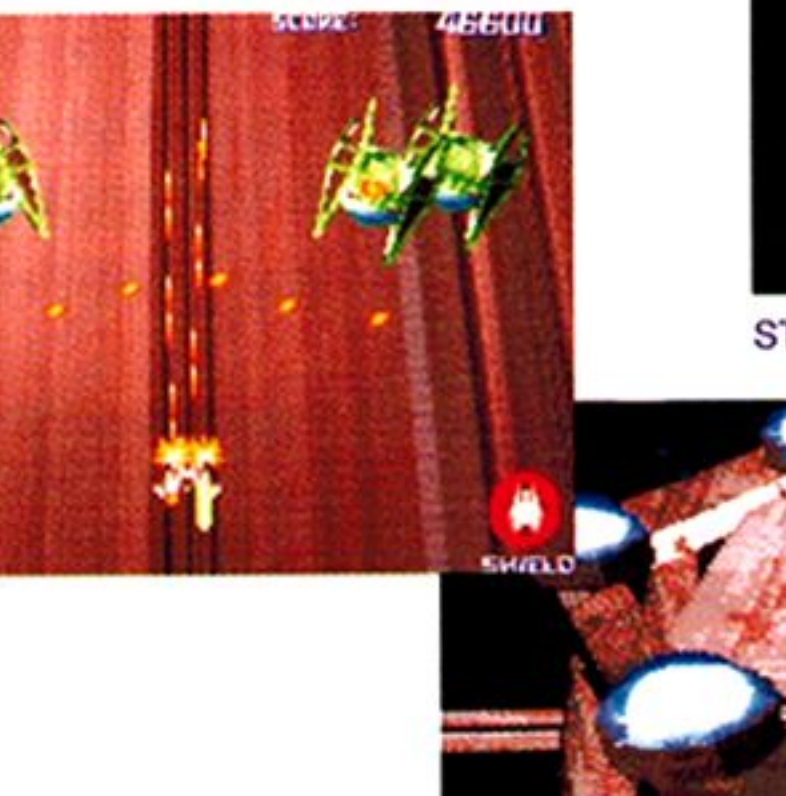
これくらいだと地上キャラとわかりやすいんだが，撃ってる暇はない

そして地上から天空へ飛び立っていく

最終面。おい，いきなり前の面のボスかよ

STAGE5。再び宇宙面だ

ワープ空間で繰り広げられる戦い。凄く綺麗なだけどね……

ワープしつつ弾をばらまくボス。一定ダメージを与えればワープしなくなる

そして最終局面へ……

Step to the Black Arts

世間ではプログラムを作るということが，非常に特別なことのように思われているのかもしれない。パソコンを使ってなにかをするというときに，それがプログラミングである確率はどの程度のものなのだろうか？
言語処理プログラム（インタプリタやコンパイラなど）がアプリの一種であって，全アプリケーションに対する製品の割合で比較すると確かに限りなく低いものになる。開発環境で新製品なんてめったに出てくるものではない。
しかし，こういった開発ツール以外のアプリを使う場合を考えてみよう。データに対して一定の処理を施していく。行き当たりばったりでやっていく人もいれば段取りをつける人もいるだろう。なんらかの段取りをつけるというのはすでにプログラミングと変わらない。一定の法則でなんらかのことをやる場合なら，できるだけ自動化しようと考えるのは当然だろう。それを行うシステムに特化しているかそうでないかだけで，本質的な違いはないのかもしれない。CGだってそうかもしれないし，打ち込み系のシーケンサソフトなどでやっていることはプログラミングとほとんど変わらない。
プログラミング言語はよりCPUに直接的な指示ができる。アプリのマクロなどだと，より高機能な記述ができる。違いは言語の「高級度」でしかないのかもしれない。
さて，現在ではさまざまなプログラム言語があり，同じ言語でも処理系が違ったりプラットフォームが違えば扱い方も変わってきたりする。
いろんなものを統一的に扱おうとするアプローチもないわけではないのだが，こういうのは簡単にはコンセンサスが取れないものと相場は決まっている。
今回多くのページを割いたTcl/Tkはなかなか面白いアプローチといえるだろう。GUIプログラムは面倒だ。実際，X Windowユーザーでわざわざ GUIプログラムを作ろうという者は非常に少ない。ちゃんとしたGUI環境と，かなり強力なウィジェットなどが完備しているにも関わらずだ。そこでプログラム部分とGUI部分を分離した形式で扱うことで，既存（？）言語のスタイルのままGUIが扱えるようにするわけだ。
これは古くからのOh!X読者の方なら，SX-BASICとウィンドウエンジンのような関係だと思えばいいだろう。言語処理系本体とグラフィカルインタフェイスを分離して扱うというのは同じ発想だ。
言語部分はなんでも構わない。高級な処理と低級な処理を切り離しつつ，「組み合わせ」という形態で統合されたシステムなのでインタフェイスさえあれば言語部分は問われないのだ。となればPerl/Tkはともかく，C/Tkというのがなんで出てないのか逆に不思議な気もしないではない。
ツールキット部分だって固定して考えることはない。たとえば，SX-BASICでは，ウィンドウエンジンの代わりにピコピコエンジンとやり取りを行えば簡単にゲーム作成もできた。フィールド型RPG用のエンジンも作られた。というより，あらゆるアプリがなんらかのエンジンとして機能するという発想だったのだが。
なによりもTcl/Tkはフリーで配布され，すでにマルチプラットフォームで展開しているというのは素晴らしいことだ。GUIに対応したマルチプラットフォームなフリー開発環境といえば，これまでは極端に機能制限の多いJavaScriptくらいしかなかったのだから。

Level1　プログラミング入門編 ……………………………80
Level2　応用プログラミング入門編 ……………………240
Level3　実践的プログラミング編 ………………………284

VBでテキストエディタを作ってみる

中野修一　Nakano Shuichi

VB5CCEを使っていろんなツールをプログラミングということで，今回はテキストエディタを制作してみます。手に馴染まないと不便きわまりないツールだけに，自分で作ればどんな機能も思いのまま。市販品やシェアウェアなどで飽き足らない人は参考にしてみてください。

テキストコントロールを使ってみる

VisualBasic（以下，VB）のプログラミングは簡単だ。BASIC自体，そう難解な言語ではないし，多彩なコンポーネントをちょちょいとやれば簡単にGUIプログラムができる。Windowsに関わる部分に踏み出さなければ，そう難しい話も出てこない。しかし，コンポーネントの枠を超えたところを狙おうとすると途端に手も足も出なくなる。いや，手を出そうと思えばできるのだが，その前にやっておくべきことはいくらでもあるだろう。

やりたいことにぴったりのコンポーネントがなかったとしても「こんなこともできない」と簡単に見切りをつけるのも早すぎる。

確かに効率のよい開発を第一とするなら，適当なコンポーネントを作ってしまうなり，適当なAPIをさっさと呼び出してやるのがいいだろう。BASICに適したところだけBASICで書いて，面倒なところはそれなりのものを使う方法だ。そもそもはそういう開発姿勢が正しいのだろう。コンポーネントにも，本当に基本的な使い方以上のものは期待せず，必要なものは別途用意してBASICはその制御のみに使う。うむ，賢いやり方だ。

しかし，これはかなりスキルと労力を必要とする開発スタイルだ。それに他人の都合の範囲内でいろいろやっていくことにも慣れないと世の中渡っていきにくいことも多い。仕事で使うとか，使用頻度が高いものだとクオリティ最重視なので，最終的に「楽をするためならどんな苦労もいとわない」というのもありだろうが，趣味でプログラムを作っている分には，「苦労をするくらいなら楽をしなくてもいいや」と考える人がいてもしかたないだろう。私もどちらかといえばその口だ。

そもそもがWindowsやVBでの開発なんて，他人の決めた仕様のなかでもがくことにほかならない。こういうのはパズルに似ている。与えられた条件内でいろいろやってみると，意外な使い方なども見えてくるものだ。いろいろ使ってみないことにはなにができて，なにができないのかなどはわからない。

今回はテキストコントロールを使ってみよう。なにをするかというとテキストエディタである。

まず，VBなりVB5CCEを起動して，テキストコントロールをフォームの上に張りつけて実行してみよう。プロパティでMultilineだけTrueにしておけば，なにもコードを書かなくても，テキストボックス内ではごく当たり前の文字入力や編集処理ができる。ちゃんとメモ帳程度のものは動く。が，それで「エディタです」などといってると本を閉じてしまう人が続出するのでもう少しだけマシなものにしてみよう。

Windowsのメモ帳とかを使っていて不満を持たない人はたぶんいないだろうし，WordPadを使っていて不満を感じない人はもっと少ないだろう。私はED.Xの人だったので，「F1で先頭行，F2で最終行に移動，Ctrl＋Kで行末まで削除，Ctrl＋Lで貼り付け，F10で行の二重化」ができないと不便を感じる。Ctrl＋HやCtrl＋Gは当然である。キーボードマクロも使えないと不便なのだが，まあこれは置いておこう。どうせX68000ユーザーでないとわからない話題だ。

それはともかく，キーの再設定や自由なコントロール処理は必須の機能だ。たとえば，標準のテキストコントロールと同様に使え，キーバインドが自由で，編集機能も豊富なものがあったとしたら……。今回はこの線で行ってみたい。

処理の流れを考える

まず，いろんな処理をしたければ，前述のとおり真っ先にMultilineのプロパティをTrueにしてやるべきだろう。これをやらないと1行しか表示できない。テキストコントロールのデザイン時に行っておいてもいいし，プログラムの開始時にプロパティ変更してもいい。ここではデザイン時に済ませている。

フォントの指定なども好みでやっておけばいいが，たいていのプログラムではどうせ動作に影響しない。テキストの一部だけ色を変えるようなことができないので，あまり使い道はない。そのほかでは用途からしてスクロールバーはつけておいたほうがいいだろう。

ざっとこんなもんだ。これだけ設定してあれば，十分使いものになる。

すでに，テキストコントロールの抱える文字列へのアクセス法は述べたので，理屈ではなんでもできるはずだ。ではテキストエディタに向けて突き進んでいこう。

まず，どういう風に処理を進めていけばいいのかを考えよう。テキストエディタは起動後，なにかキー入力があったら文字表示，ないしは，それに応じた動作を行う。ということで，メイン部分は実に単純だ。

```
キー入力された文字コードを取る
入力に応じた処理
```

この連続だ。普通の文字が打ち込まれたら，それをカーソル位置に表示し，コントロールコードが打ち込まれたら，それに応じた処理をする。実際にはこれにいくつかのメニュー処理などが割り込んでくるのだが，基本は変わらない。普通の文字がそのまま入るという処理はとりあえずテキストコントロール素通しで構わないだろう。コントロールコードはほとんど処理されないので，処理を奪ってやらなければならない。これはKeyCodeごとに処理を分ければ済む話だ。

キー入力が行われると，KeyDownイベントが発生し，処理ルーチンがあればそこへ飛ばされる（Text1_KeyDown）。これはVBの基本動作だ。ありがたくそのまんま処理を進めよう。

入力された文字がコントロールコードかどうか，コードで判定し，各種キーに対応した処理ルーチンへ飛ばしてやる。

ただし，もともとのテキストコントロールでサポートされているコントロ

ールコードだとちょっと面倒なことになる場合もある。アタマ跳ねで処理を奪ってやっても，そのあとでVB自前の処理が行われてしまう。別にそれでもよい場合もあるだろうが，マズイ場合もある。そこで，パラメータとして渡ってきたKeyCodeの値を書き換えておいてやろう。値渡しか参照渡しかとか，ちょっと面倒な話もあるんだが，ここは素直に書き換えておくだけでいい。処理が少し進んでからだと書き換えても間に合わないみたいだが，KeyDownに飛び込んですぐなら有効だ。ここではKeyCodeを0に書き換えている。

例としてカーソルキーの上下を見てみよう。こんなものは標準のテキストコントロールでも当然上を押せば上，下を押せば下にカーソルが動くようにできている。動作はほとんどメモ帳と同じなので，なにか適当なテキストファイルで確認してみるといいだろう。

このままではちょっと使えないので，カーソル位置保存の動作に変更する(好みの問題だ。昔はオリジナルED.Xのままのカーソル位置非保存でよかったのだが，SuperED以降，カーソル位置保存のほうに慣れてしまった。以前は全然不便ではなかったのだが……)。

カーソルを上に持っていくときには，まず現在のカーソルカラム位置を数えて，ひとつ上の行の同じ位置に移動させればいい。実はCD-ROMに入っているプログラムはバグが残っているのだが，なんのことはなくてもかなり苦労させられた部分だ。まず，「カーソル位置」と簡単に書いたけど，VBではカーソル位置を得る関数などはないのだ(私は見つけられなかった)。

Selstartが勝利の鍵だ

根本的なところに立ち返ってしまうのだが，今回は「テキストコントロールを使う」と題して話を進めてきた。なにをするかというと「テキストエディタを作る」だ。処理内容は，「キー入力を得て，文字を表示する」だ。

とはいうものの，これだとテキストコントロールを使わずに文字表示とキー入力ができる，もっと軽そうなコントロールでもいいんじゃないかという感じもする。それでもできなくはないだろう。これは根源的な疑問だと思う。

かといって，せっかくひととおりの動作をするテキストコントロールを殺してしまうのももったいない。なにからなにまでBASICで記述するというのは速度的にもちょっと問題があるだろう。そこで，現状の機能はできるだけ生かしつつ，必要な機能を加えていくことにする。やはりテキストコントロールだ。

いまさらだが，テキストコントロールで使用可能なイベント，プロパティ，メソッドを一覧してみよう。ほら，こんなに高機能そうじゃないか。

ここで重要な問題になってくるのが前述のとおり，カーソルコントロールだ。テキストコントロールで作業すると，当然カーソルは表示されるのだが，これを直接制御する方法は用意されていないように見える。

自前でカーソルを作るという手もあるが，位置や表示を全部自分で管理しなければならない。キー入力は全部把握できるので，やってできないことはないが，マウスで「ちょい」っとやられたときは，対応する位置にカーソルを移動させねばならない。これはなんとなく面倒そうだ(復刊号のダンプツールではその処理を当たり前のようにやってたんだけど，これは気分の問題)。

あるものはなるべく利用する，無駄に新しいコードを書く必要はないだろう。が，いまになって思えば，自前でカーソル表示をしてればずいぶん楽だったのかもしれないなあ……。

問題は現在のカーソル位置を直接取ってくるプロパティがないことだ。オブジェクトブラウザでテキストコントロールを眺めて使えそうなものを探す。そこで見つけたのがSelstartだ。これは範囲選択時の先頭位置を返すものだ。範囲選択してないときは，これから範囲選択が始まる位置を返してくる。さらに，

```
Text1.Selstart = n
```

とすれば，カーソル位置も移動する。

これは楽勝！　と処理を進めたのだが，やがて大きく行き詰まることになる。

カーソル位置を得る

念のためにいっておくと，Selstartで返ってくるのはカーソル座標そのものではない。Text1.Textという大きな文字列のなかで先頭から何文字目かという数値を得ることになる。

COLUMN　テキストコントロールの基礎

テキストコントロール（Text1という名前だとしよう）の編集している文書は，Text1.Textで参照できる。これは文字列としてそのまま扱える。

```
Text1.Text = work$
```

とすれば内容の更新も簡単だ。

確認しておくと，テキストコントロールに入力されているすべての文字列はひとつの文字列として管理されている。たとえ何百行あろうと，ひとつの文字列だ。

BASICの文字列処理はそれなりに柔軟なので，いっぱしのBASICプログラマなら，その気になればなんでもできるだろう。ちなみに文字列の長さは2GB分までサポートされている。CPUのアドレス空間を全部16進ダンプしたいとかいう向きの方は，しかたがないので文字型変数を4つ使ってやってほしい。

テキストコントロールへのキー入力は，Text1.Keydownイベント処理ルーチンでのKeyCodeで与えられる(KeyCodeという変数をアクセスすればそこに入っている)。となれば，必要な情報はすべて揃っていることがわかる。

8ビット時代からやってる人にはこれだけで十分だろうが，一応，入門編なので念のために文字列処理についてまとめておこう。

●変数宣言

まず，文字列型の変数を宣言する。先頭の宣言部で，

```
dim test1 as string
```

のように書き込む。test1という名前の文字列型変数ができた。

この辺りの変数の型はいろんな言語でほぼ似たものが使われる。それだけにまぎらわしいこともある。しかし，strなのかstringsなのかと思い悩むことはない。打っていればVBが自動的に補完してくれるのだ。

●文字列の代入

文字型として宣言された変数には，

```
a$="This is a pen."
```

のように文字列を代入することができる。VBにはvariant型というどんな型にでもマッチできる変数型も用意されているので，どうしてもダメなもの以外はそれだけで押し通すこともできなくはないのだろうが，できるだけ自動処理をさせないほうが開発効率を上げることができる。

先ほどのような文字列の補完処理くらいならいいのだが，プログラムの中身では自動処理されるものは少ないほうがいい。できるだけ明示的に処理を記述しよう。プログラムする側がきちんと把握できなければ意味はない。また，思わぬところでのエラーを未然に防ぐこともできるだろう。

●文字列の切り出しと結合

文字列の操作は，文字列内の文字の位置を指定して任意の文字を取り出したり，くっつけることができる。

文字列の切り出しにもっとも多用されるのはmid関数だ。

```
mid(str1, 3, 5)
```

の場合，文字列str1の3文字目から5文字分を抜き出してくる。文字列の長さは，

```
len(str1)
```

などのようにして求めることができる。len関数も文字列処理では多用される。

文字列の結合は単に，足し算してやればいい。

```
str1 + str2
```

●文字の判定

取り出した文字列の中身がどうなっているかを調べるには，1文字ずつ取り出して文字コードを調べることになる。文字からコード(数値)を得るためのものがasc関数で，逆に数値からそのコードに対応した文字を作るのがchr関数だ。

```
asc("A")
chr(65)
```

でそれぞれ，65，"A"を返す。文字列の先頭から何文字，末尾から何文字という風に取るleft()，right()もよく使われる。

文字列同士の比較はif文を使って，

```
if str1 ="test test test" then ～
```

のようにする。

さあ，これで文字列処理は完璧だ(？)。わからないことがあったらメニューからヘルプを選び，項目を探してみよう。

AddItem	Font	LinkClose	MouseUp	RightToLeft
Alignment	FontBold	LinkError	Move	ScrollBars
Appearance	FontItalic	LinkExecute	MultiLine	SelLength
BackColor	FontName	LinkItem	Name	SelStart
BorderStyle	FontSize	LinkMode	OLECompleteDrag	SelText
Change	FontStrikethru	LinkNotify	OLEDrag	SetFocus
Clear	FontUnderline	LinkOpen	OLEDragDrop	ShowWhatsThis
Click	ForeColor	LinkPoke	OLEDragMode	TabIndex
Container	GotFocus	LinkRequest	OLEDragOver	TabStop
DataChanged	Height	LinkSend	OLEDropMode	Tag
DataField	HelpContextID	LinkTimeout	OLEGiveFeedback	Text
DblClick	HideSelection	LinkTopic	OLESetData	ToolTipText
_Default	hWnd	Locked	OLEStartDrag	Top
Drag	IMEMode	LostFocus	Parent	Visible
DragDrop	Index	MaxLength	PasswordChar	WhatsThisHelpID
DragIcon	KeyDown	MouseDown	Refresh	Width
DragMode	KeyPress	MouseIcon	RemoveItem	ZOrder
DragOver	KeyUp	MouseMove		
Enabled	Left	MousePointer		

図1　テキストコントロールのサポート機能一覧

簡単なサンプルプログラムを書いてみよう。Multilineのテキストコントロールを置いて，

```
Private Sub Text1_Key_Down(KeyCode)
Debug.Print Text1. Selstart
End Sub
```

のような簡単なプログラムを作ってみてほしい。

実行してなんやかんや入力して，いろいろカーソルを動かしてみると，ときどき「？」な動作をすることがわかる。カーソルの移動方向が変わったときなどに，それを無視したような挙動を示す。

カーソルを1文字ずつ右方向に移動すると，文字位置は，

```
4 5 6 7
```

などという風にインクリメントされる。これはいい。ここでカーソル左で1文字戻すと，

```
4 5 6 7 8
```

と6の位置のカーソル表示なのに，Selstartは8を返すという感じだ。選択位置の先頭というよくわからない基準で値が返ってくるわけだ。カーソル上下も絡むとなかなか一筋縄でいきそうにない感じだ。

文字列の入力方向を覚えておき，キーコードごとにテーブル化して対応……とも思ったが，見れば見るほどになかなか半端ではない。そんな大げさなことをするくらいなら，全部自前で管理したほうが楽ではないか？　と，悩ましく，実に曲者である。

で，ここでキー入力後にシフトキーをちょんと押してみてほしい。表示されるカーソル位置の値が瞬時に正しく補正されることがわかる。今回はこの手で対応することにした（ここで素直にあきらめて自前で処理しておけば何日か楽できたのかもしれないなあ……）。

SendKeysでキー入力

キーを押したことにさせるにはどうすればいいのだろうか？　入力されたKeyCodeをすぐさま書き換えると，ある程度対応できそうなのはすでに述べたが，今回はこの手は使えない。VBにはちゃんとキー入力をさせる命令があるのでそれを使おう。それがSendKeysである。残念ながらシフトキーを単独で押すという動作は送れないみたいなので，しかたなく普段使いそうになくて害のなさそうなキーを代わりに押してみる。今回選んだのはF16（ファンクションキーの16番目）である。たいていのキーボードではF12までしかないと思うので，直接押すことはまずないだろう。このコードを送ってみると，キー位置の補正もちゃんと行われているようである。

ただ，SendKeysによる人工のキー入力も同じ処理ルーチンを通るので，この入力のあとには同じキー入力補正は必要ない。適切にスルーしてやらないとループにはまってしまう。なお，間違えてはまったときは，シフト＋Pauseでブレイクするので，VBプログラムで無限ループに入ったときの対処法として覚えておこう（AT互換機の場合。PC-98でもBREAKと書いてあるキーでいいはず）。どうやって区別するかというと，わざわざ普段押せないキーを選んでいるのだから，KeyCodeでそのまま判別すればよい。

試してみると一応動く。問題ない。キー入力のたびにNumLockキーが点滅してる気がするが，実害は少ししかない。とりあえず気にしないことにしよう。

各種機能を作る

これでようやくカーソル位置が手に入った。カーソル位置への処理をするなら，Selstartで得た値の位置までポインタを進めて……というと語弊があるので，ちゃんと書こう。文字列をペーストするなら，

```
right(Text1.Text, Text1.Selstart)
```

で得た文字列と，

```
left(Text1.Text, len(Text1.Text)－Text1.Selstart)
```

で得た文字列のあいだにほかの文字列を挟み込んだり（クリップボード系の処理を使ったほうが楽だが，ED.Xは非クリップボード系のカットバッファを持っていて便利だった），この場所からいろんな処理を進めていったりすることになる。

なにをやればいいのか？　それはコントロールキーなりエスケープキーでのやりたい処理内容による。そこは自分でプログラムするわけだ。

「Ctrl＋Pで行末へ」であれば，カーソル位置から後ろ方向に文字列をサーチしていって，改行コードを見つける。見つけたら，そこの値でSelstartを書き直してやればいい。改行コードはお馴染みな人にはお馴染みな，13，10（^M^J）の並びだ。改行したいときにはこのコードを文字列化して挟み込めばいい。

少し脱線するけど，上記のようにアイテムのプロパティを変数代わりに頻繁に参照するのはあまりおすすめしない。実行速度がガクっと落ちるからだ。多用するものは変数に取って使うと遙かに高速化できる。が，こちらの意図してないタイミングで値が変わってたりするものがあると，かなり深刻な状況になる。値が変わるのか？　変わるのである。単に全体の動きを把握してないのが悪いだけなのだが，あちこちの副作用とか，イベント駆動なので処理がどう流れてきたのかをとらえにくいこともある。テキストに値を代入したらスクロールバー値が変わってたとか，気がつかないとかなり

ハマることになる。

脱線が続いているので話を元に戻そう。実はカーソルを上に上げるという話だ。こうやって苦労して得たカーソル位置は，カーソルのXY座標(columns，lines)というわけではないというのはすでに述べた。しかしカーソルを上に持っていくには，何行目の何桁目にあるのかが重要になる。必要なんだから，求めてやればいい。

文字列の現在位置から，前方に向かって文字を切り出しつつ改行コードを探す。その改行コードの位置との差が，現在の桁位置になる。さらにもう1回前方に改行コードを探し，その位置から必要な桁数分だけ後戻りした値にSelstartを書き換えればようやく上にカーソルが移動することになる。

上の行の文字列が所定の桁数に至らない場合は末尾で止めて，桁位置だけは保存してさらに上の行を見るときに使ってやる。途中でTABコードがあったら，8文字単位(暫定TAB文字数)にアラインメントして調整する。そのほかいちいち挙げないが，あちこちで必要になるエラー処理を組み込む。

こうして初めてカーソルが上の上移動が完成する。カーソル上下に比べれば，ほかのたいていの処理はかなり単純だ。あと複雑そうなのは，1ワード前後に移動するとかいうカーソル制御だろう。単に区切り文字をサーチすればいいだけだが，日本語文書で適切に動いてほしいと思ったりすると，やや複雑になる。停止条件が単に増えるだけなのだが。どう動いてほしいか，というのと，理屈さえわかればなんでもないことではある。

以下同様に個別処理する。ED.Xのコントロールコードはほとんど組み込んだ。1つひとつは非常に単純なので，根気だけの問題である。一部，ダイアモンドキー系は私は使わないので実装していない部分もある。Ctrl+C，Ctrl+X，Ctrl+VにCtrl+Zだ。これらはWindows標準のコントロールを素通しさせたほうが使い勝手がよくなると思っている。変なことをやっても，なにもプログラムしてないのにCtrl+Zが効いたりするのは感動ものだ。いや，自前でUndoを実装するのはしんどいこともありそうなんで。

こうしてキー操作部ができたら，次はファイル入出力なども付け加えてみよう。なんやかんや自分でやるよりは，素直にコモンダイアログを使うのが圧倒的に効率がいいのは前回も話したとおりだ。ちょこっちょこっといじって，Textに設定する文字列をファイル処理するだけの話だ。簡単。テキスト入力部を標準のテキストコントロール互換にしたいときは，対応するプロパティやイベント処理部分を全部つけないといけない。

ざっと入れてみると，意外とちゃんと動くことがわかる。最近のマシンはかなりのスペックだし，VBでもそれなりのことはできそうだ。

しかし，いろいろテストしてみると，独特の不具合も出る。まず，簡単なところで，TABの処理だ。WindowsではTABキーを押すとフォーカス(現在の処理を行う場所)が次のアイテムに移ってしまう。これはキー入力を横取りする前に処理されてしまうので，このキー入力ルーチンでは処理できない。で，どうするかというと，ダミーのアイテムを作成して，そちらにフォーカスが移ったら，テキストコントロールのほうにフォーカスを戻してやる。GetFocusというイベントが発生するので，その処理ルーチンを記述すればいい。

現在困っているのは，コントロールキーのほうだ。このテキストエディタは，テキスト入力部だけ独立したコントロールとして作り，それをファイル制御などを担当するエディタコントロールに貼り付けて使うということを前提としていた。そのエディタコントロールをなにかに貼り付ければできあがりという具合だ。要するに最初から全体をコントロールとして作成していたのだが，実際にHTMLページに貼り付けると，いくつかのコントロールキーについてはIEが先にキーを取り込んでしまい，こっちに渡してくれない……。IE5CCEでの制作途中では，コントロール状態での実行でもそういうのはなかったので全然想定していなかった。実行環境でのデバッグは重要だなぁ……。さすがにこれは手の出しようがない。ビルぅ〜。

コントロールでいろんな機能を作成して，HTMLで呼び出し，アクティブデスクトップとかツールバーにビルトインしつつ環境を再構成していくという構想だったのだけど，根本的に見直しが必要かも……。

ということで，これは開発環境上のままか，VB5/6の実行ファイル形式で使うしかない。これを読んでる人はVBかVB5CCEくらいは持ってるよね？ ……怪しいか。VB6の体験版があるということだったので，CD-ROM収録をお願いしたのだけど，ムックだとダメだそうで，とにかくVB5CCEはみんな入手しておこう。期限切れでVB5CCEがCD-ROMに収録できない状況は前回と変わらないが，Webでの配布は現在も続けられている。さっさとVB6CCE作りゃいいものを。まったく。

最後に

いや，最初の頃はSelstartの制御法に気づかず，そのまま辻褄をあわせながら処理を書いていったので，あとで大変なことになってしまった。書き直す暇がなかったので，現状でも構造はねじくれたままになってしまっている。Selstartを使ってるにも関わらず，範囲指定は別途できなきゃなあとか(Selstartは保存して，Selsizeでカーソル位置を変える。前方移動はさらに面倒)，そういうのを実装しかけてから根本的な変更になったので挙動が把握できなくなっている部分がある。ああ，説明だけはしておいたので，エディタを作りたい人はちゃんと作り直したほうがいいかも。いや，作り直すなら，もうカーソル位置は自前で管理かなあ……。

さて，今回はダミーのキー入力を入れてカーソル問題を解決したわけだが，実害がちょっとだけあると書いた。ちょっと長い文書を読み込んで試せばすぐにわかるだろう。カーソル右，

→→→→→→→→→→→66→→→

あれ？ NumLockしてテンキー部を使ってるんなら納得しなくもないが，カーソルキーでなんで対応するテンキーの数字が？ 実際に入力されたキーコードをそのまま取り出してみても，当然ながら数字キーのコードが入っている。……これって，私のせいじゃないよね？

やっぱり作り直しかなあ……。

＊　＊　＊

その昔，OS-9ユーザーはみんなテキストエディタを作っていたという。誰もがみんなエディタばかり作る。なんでエディタを作るのか？ それはそれを使って次のエディタを作るため，だそうだ。エディタはあらゆる作業の要，結構いいところまではVBでもいけるということがわかった。もう一息だ。も少し精進だな。

図2
今回のフォームの構成。
3段構造になっている

図3 バグ……

Future BASICによるMacintoshプログラム制作第2回
画像の表示とアニメーション

古籏一浩 Furuhata Kazuhiro

前回はFuture BASICを使って時計を作成しましたが，今度は画像を表示したり，アニメーションさせてみましょう。今回はFuture BASICだけでなくResEdit(リソースエディタ)も使用します。

PICT画像形式

Macintoshといえばグラフィックに長けたマシンといった印象があるかと思います。Future BASICでも画像を扱うことはできます。画像の表示は手軽にできますが，画像処理やお絵書きソフトなど，ちょっとしたものを作る場合は複雑な手順が必要となります。今回は画像処理などは行わず，簡単な画像表示とアニメーション表示をさせてみましょう。

画像といっても今回扱う画像はMacintoshの標準画像形式であるPICT形式です。PICT形式はベクトル形式とビットマップ/ピクセルマップを混合して扱うことができます*1。 PICT形式はMacintoshの標準形式ですのでグラフィックを扱うソフトでは確実にサポートされています。もちろんMacOS側でも手軽に利用できるようなサービスが用意されています。PICT形式を解析して表示するとか，自分でPICT形式のコードを吐き出すような特殊な場合を除けば手軽で扱いやすい画像形式といえるでしょう*2。

まずは，PICT画像をウィンドウに表示させるところから始めましょう。

*1 実際はJPEGやEPS形式なども含めて扱うこともできます。EPS形式など特殊なもの，独自のコードを埋め込む場合はPicCommentを使ってほかのデータに影響が及ばないようにします。別ページにPICT画像形式の解説を用意しましたので参考にしてください。

*2 Macintoshが発売された当時(1984年)は強力でしたが，時代の流れとともに力不足となってしまった感じがあります。これを解消しようとしたのがQuickDrawGXですが，あえなく挫折し，事実上消滅してしまっています。まあ，せっかく作ったから，培った技術はQuickDrawに加えようという話もあるようですが……。

まずやるべきことは画像の用意

PICT画像を表示するといっても，表示すべき画像を用意しないことにはどうにもなりません。とりあえずインターネット上からダウンロードしたGIF，JPEG画像をPhotoshopなどでPICT形式に変換して保存しておきましょう。

ここでFuture BASICのマニュアルを開いてじっくり眺めてみます……が困ったことに「PICT画像を表示する」という命令はまったくありません。Macintoshの標準形式であればサポートしていてもよさそうなものですが残念ながら存在しません。PICT画像形式のファイルを読み込んで表示させるには，いくつもの手順を踏まなければならないのです*3。

BASICなんだから，もっと手軽に表示できてもよさそうなものですが，ないものはないのであきらめるしかありません。PICT形式で保存されたファイルの読み込みに関してはまたの機会にするとして今回はFuture BASICで用意されている方法で，なんとかPICT画像を表示させてみることにしましょう。

Future BASICはPICT画像形式のファイルを読み込み表示させる機能はありませんが，「PICTリソース」を表示させる機能は存在します。

*3 ハンドブックマニュアル308～309ページの「PICTファイルの保存および読み込み」を参照してください。読み込むだけで20行近くかかってしまいます。手順さえわかれば，どうということはないのですが……。

PICTリソースってなに？

PICTリソースはPICT形式ですがファイルではありません。PICTリソースを扱う前に「リソース」について説明しておきましょう。

リソースは再利用可能な部品といえます。たとえばアプリケーションを作成するにしても，ボタンやアイコン，ウィンドウ形状など共通化できる部分はたくさんあります。これらをリソース(部品)として用意しておくことで開発効率の向上とデータの共通化を行っています。

Macintoshでいうリソースはアプリケーションだけでなくファイルにも存在します。Macintoshのファイルシステムでは，ひとつのファイルに対して「データフォーク」と「リソースフォーク」の2つの格納場所を用意しています。データフォークが表側，リソースフォークが裏側といった感じになるでしょうか。HTML文書を手書きできる方であればHTMLのタグがリソースフォーク側，文章がデータフォーク側といった感じで考えてもらって

図3
新しく作成する場合はNew…，既存リソースを開くにはOpen

図1 ResEditのアイコン

図2 飛び出すピエロ
ちなみに豚もいる。option＋command＋shiftキーを押したままAbout ResEdit…を選択

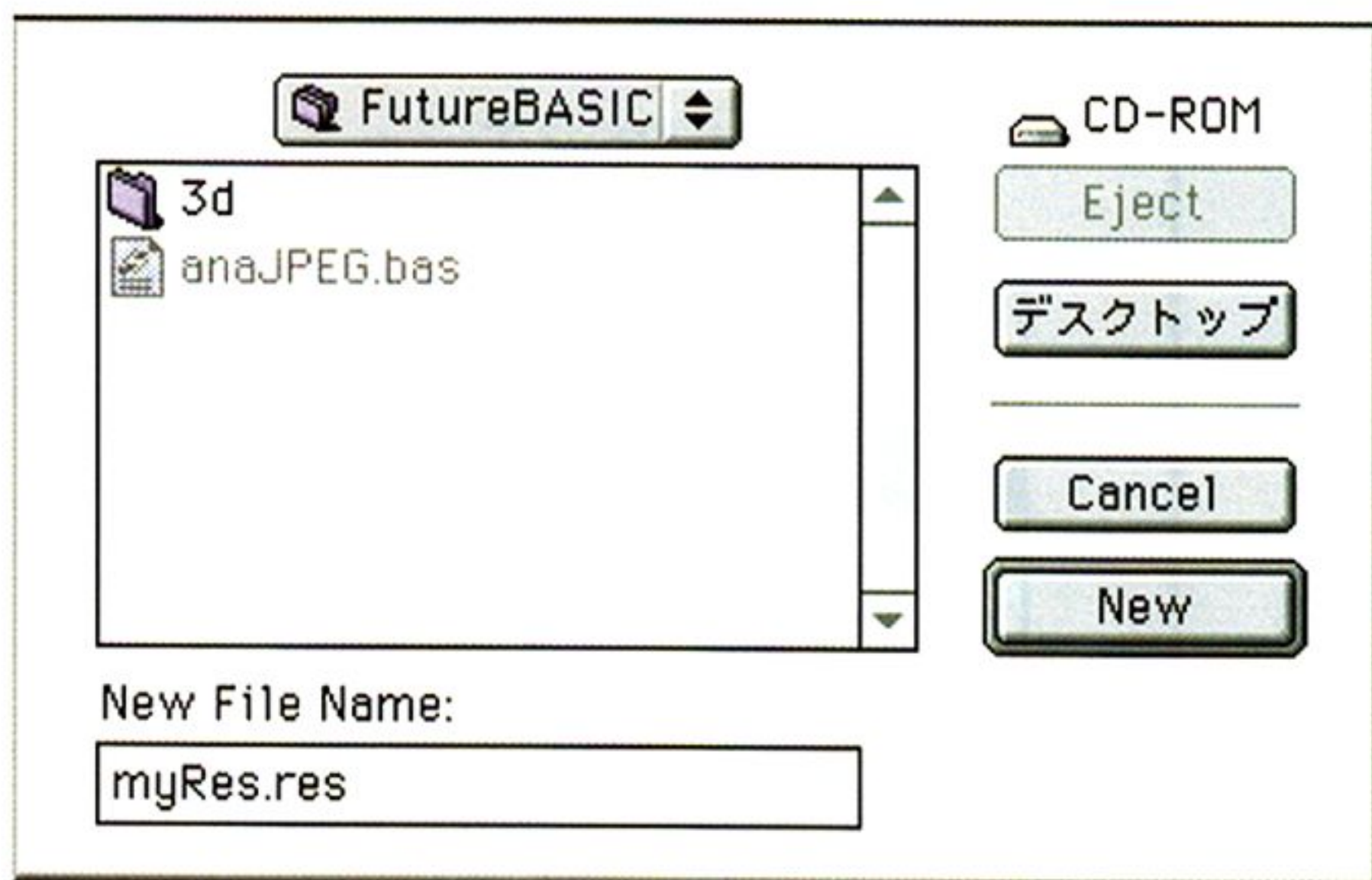



図4
作成するリソースファイル名を入力

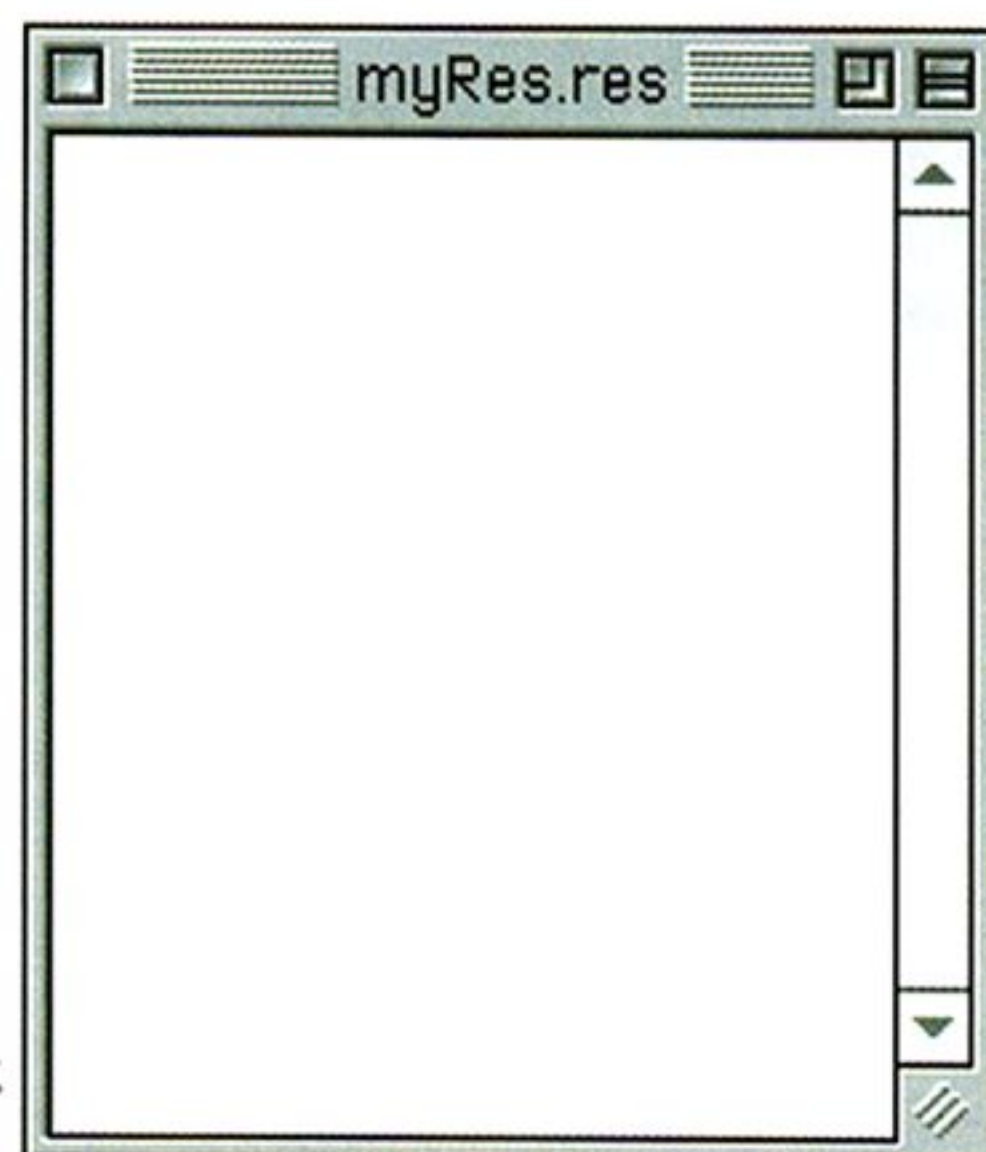



図5
現在のリソースフォークの内容。最初はなにもないのでからっぽの状態

もよいかと思います。

PICT画像ファイルの場合はデータフォーク側にPICTデータが格納されます。これに対してPICT画像リソースの場合は，リソースフォーク側に格納されます。格納される場所が違うだけで中身のデータは一緒です。

「それなら別々に分ける必要ってあまりないんじゃないの？」

もちろん用途にもよりますので「必ずしも分けなければいけない」ということはありません。基本的にはデータフォークにはほかのアプリケーションでも使用できるような共通のデータを，リソースフォークには自分で扱う固有のデータを格納するといった感じになっています。

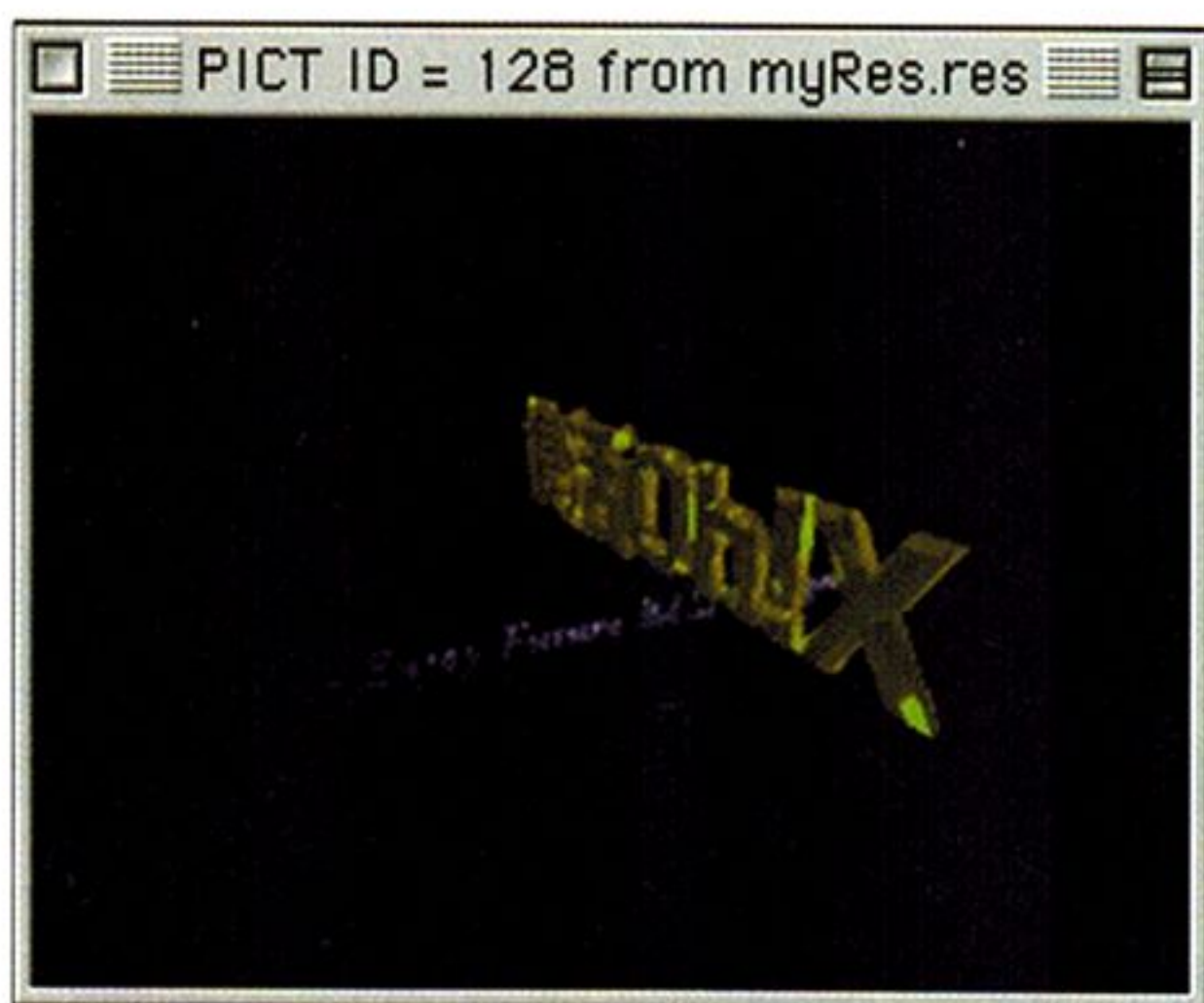
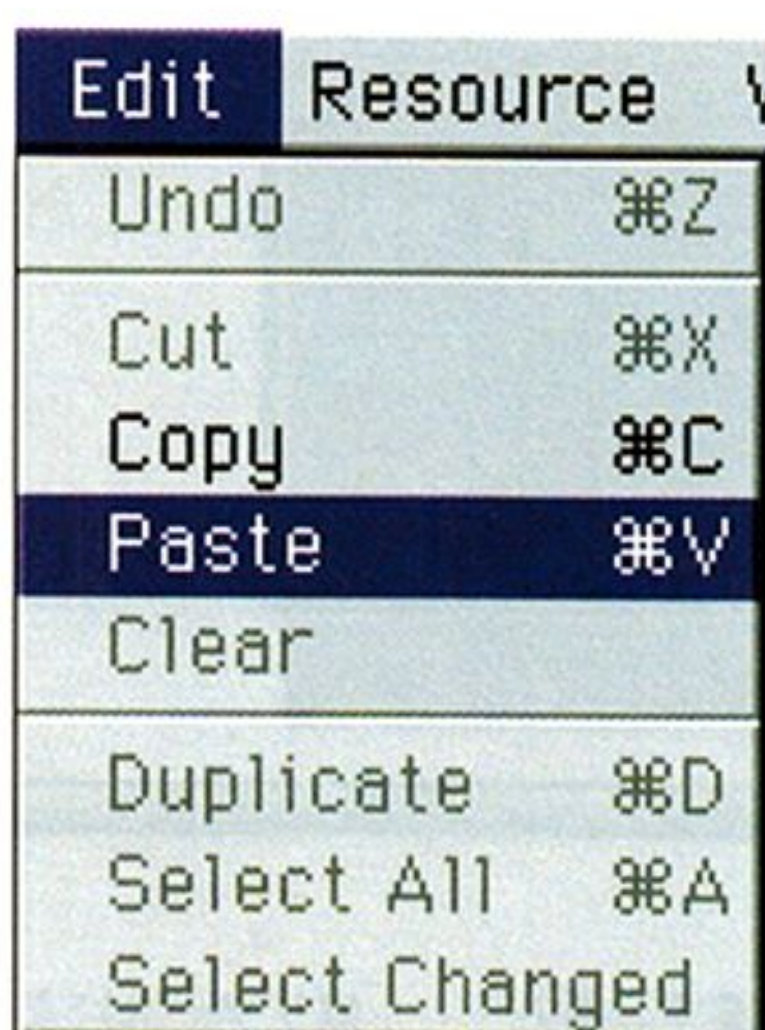
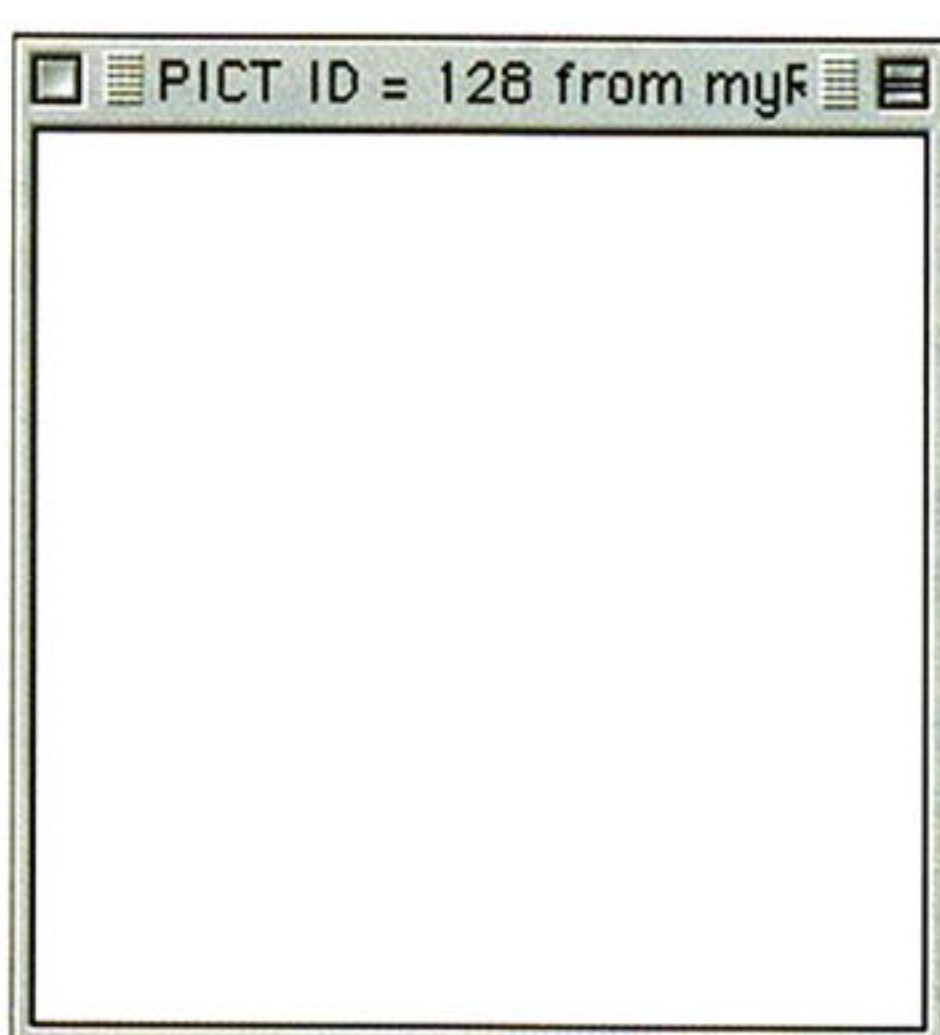
データフォークの中身を操作するのは簡単ですがリソースの場合は，どのリソースが何番で格納されているか，どういう名前で格納されているか知らないと正しく扱うことができません。たとえばワープロソフトで，どのワープロでも読めるようなテキストのみのデータをデータフォークに，文字を斜体にしたり，指定したフォントにするといったワープロ独自のデータはリソースフォークに格納しておきます。ほかのワープロなどでデータを読み込む場合，データフォークにあるテキストデータを読み込めば最低限文章だけは読み込むことができます。わざわざ変換ソフトで変換したり不要な部分を削除しなくてもよいのです[*4]。

＊4　多くのソフトではヘッダ（データの先頭部分）に文書情報を用意します。文書内でも独自のコードを埋め込むためほかのソフトでデータを読み込む場合は，作成されたアプリケーション固有の形式を知らないと表示することすら難しくなります。共通のデータと固有のデータを分けておけば変換する手間もなく単純にデータフォークだけ読み込めばよいわけです。ワープロだけでなく画像にも似たような手法が使えます。データフォークにPICT形式，リソースフォークにTIFF形式，EPS形式を用意しておけばひとつのファイルで複数の画像形式を混在させることが可能になります。

図6

図7

図8

図9

グラフィックソフトで画像をコピーしたあとペーストする。PhotoshopでPICTリソースで保存してもかまわない。数枚であれば，そのほうが楽かもしれない

PICTリソースの作り方

ここが今回のポイントです。PICTリソースを作成するには「ResEdit」（リソースエディタ）が必要です。これがない場合はPhotoshopなどPICTリソースとして保存できるグラフィックソフトを利用します。Future BASICを購入すればResEditは付属してきますので，ない場合はFuture BASICのCD-ROMからコピーしましょう。

図1がResEditのアイコンです。ダブルクリックすると図2のような起動画面が表示されます。設定によっては起動画面でなくファイルオープンダイアログが表示されることもあります。

今回は新しくリソースを作成するので「File」メニューから「New...」を選択します（図3）。すでに作成してあるリソースやアプリケーションなどのリソースを表示する場合は「File」メニューの「Open...」を選択します。

myRes.resと入力して「New」ボタンを押します。ここでつけるファイル名はなんの制限もありませんので自由につけてかまいません。できれば英数字のほうが安全です（図4）。

ボタンを押すと図5のようななにも中身のないウィンドウが1枚表示されます。今度はPhotoshopなどでPICTリソースにする画像を表示します。画像をコピー（command + C）します。コピーしたらResEditに切り替えてペーストします（図6〜9）。

これは手抜きな方法ですが，指定したリソース

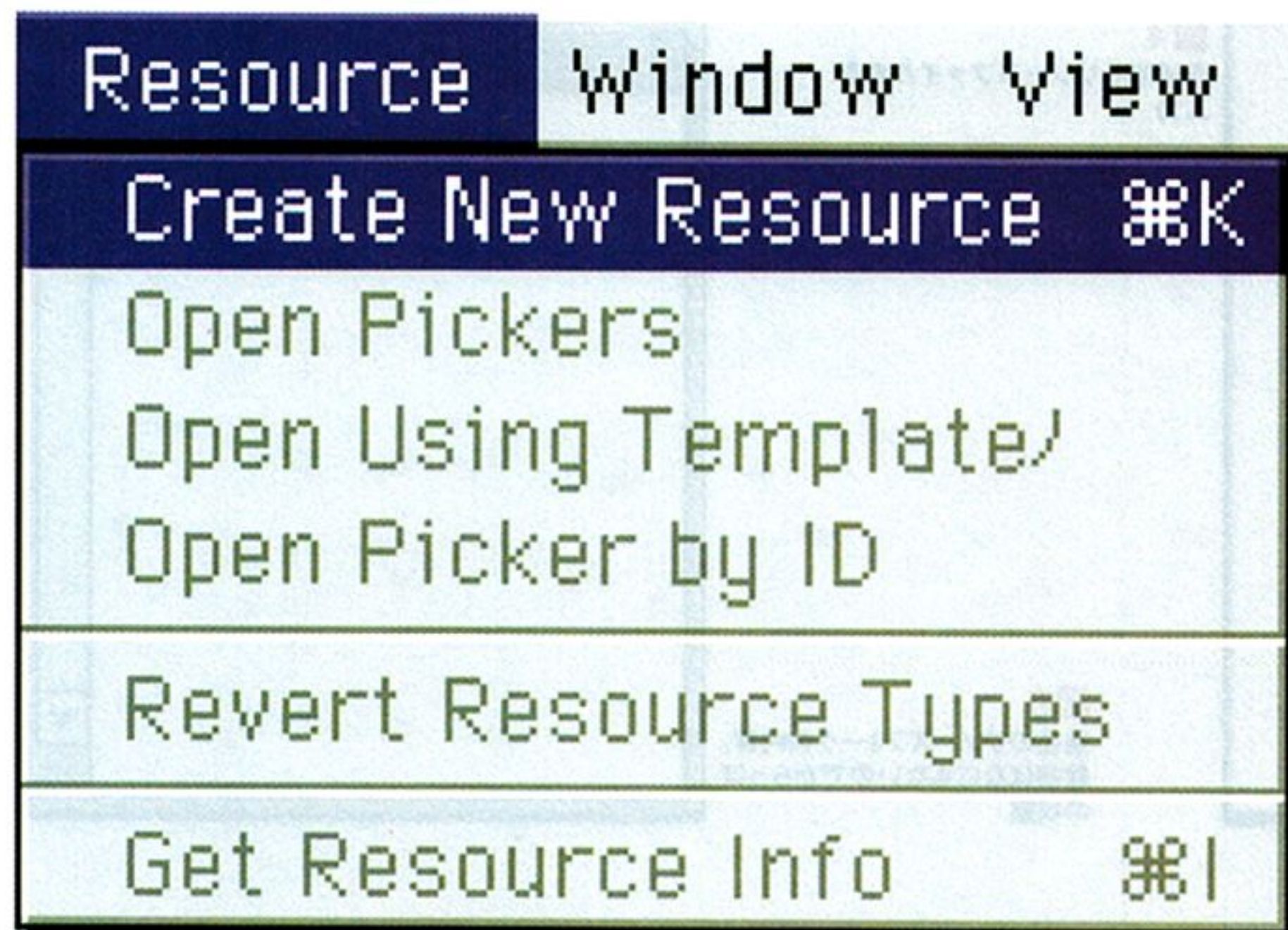



図10
新規，独自リソースを作成するにはResourceメニューからCreate New Resourceを選択する

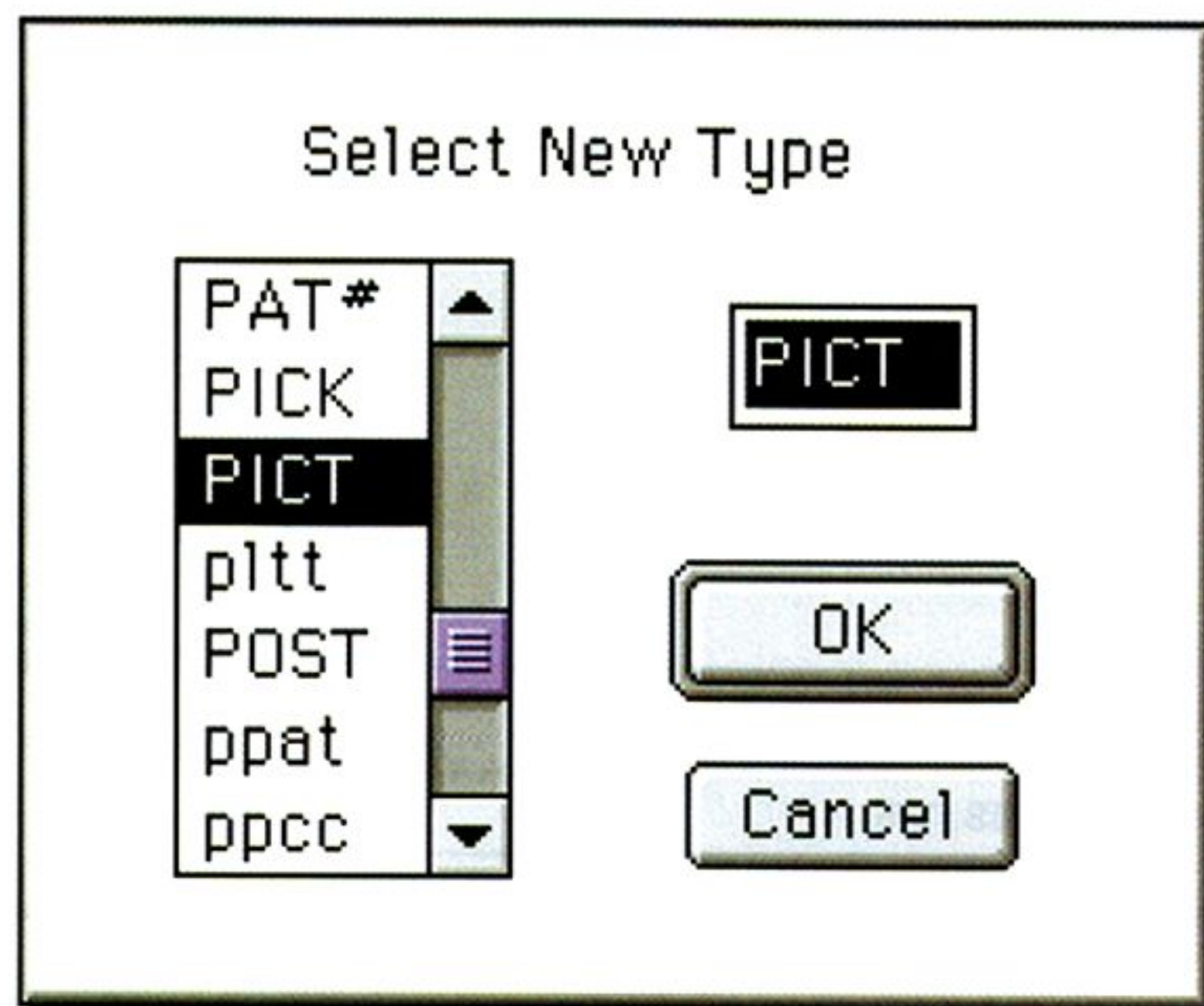



図11
ここで該当するリソースを選択するか，存在しない場合は4文字でリソースタイプを入力する

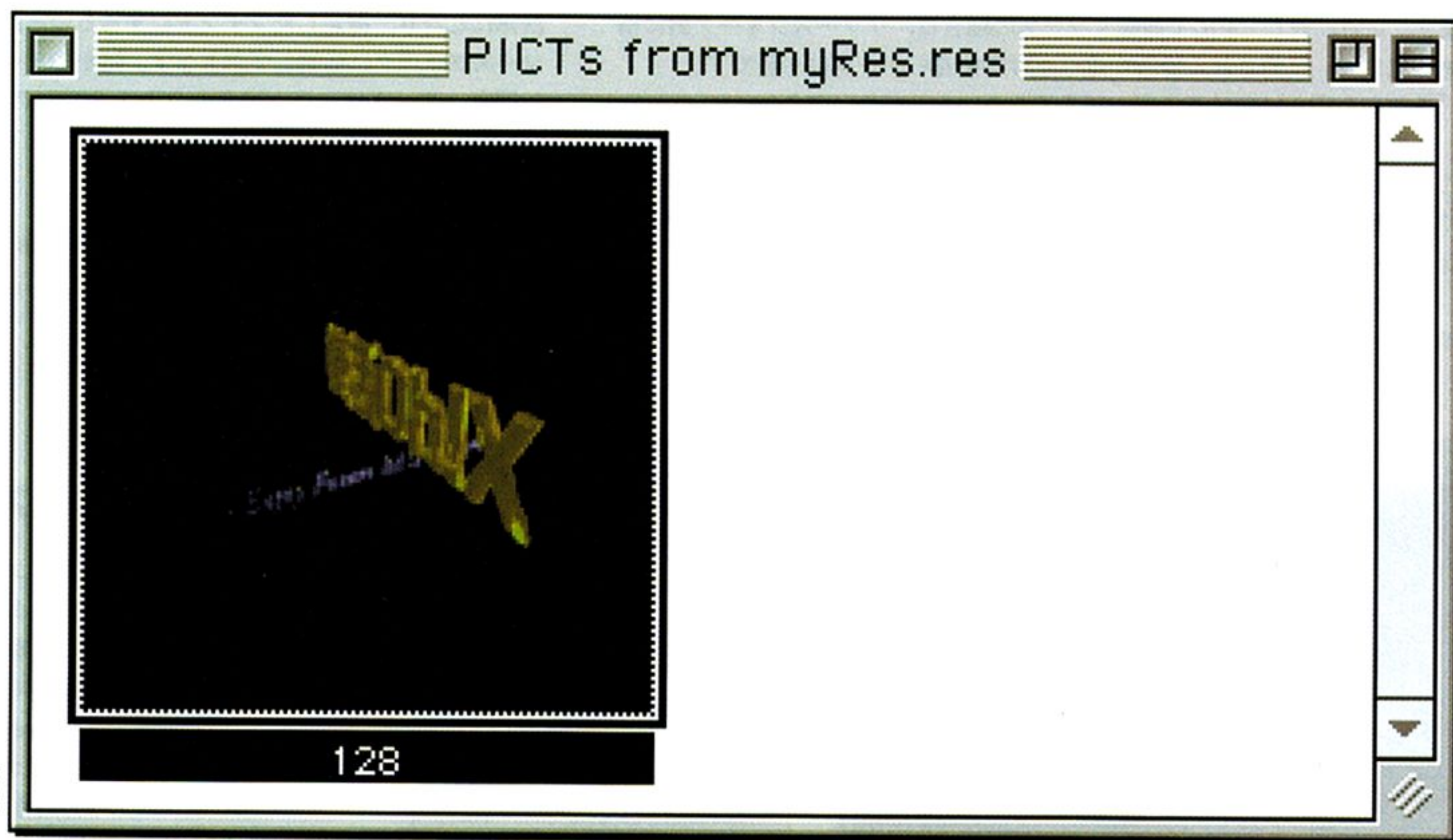



図12
128番のPICTリソースができた。再度ペーストすると今度は129番のPICTリソースが作成される

(ここではPICT)を作成するには図10，11のような手順を踏みます。新しくリソースを作成する場合や，独自のリソースを作成する場合，「Resource」メニューから「Create New Resource」を選択し作成するリソースタイプを選択します。

もし存在しないリソースタイプもしくは独自のリソースである場合は右上の部分にMZ70などと入力します。リソースタイプで使用する文字数は「必ず4文字」と決まっていますから，ABCやBASICなど4文字以外のリソースは作成することができません。

ウィンドウを図12のように128番(ID=128)のPICTリソースが作成されているはずです。もし，作成されない場合は図6の手順から再度やってみてください。

各リソースに含まれるものはリソースIDという番号で管理されています。番号で管理するだけではわかりにくい場合が多々あります。そのためリソースは番号だけでなく各リソースに名前もつけることができるようになっています。リソースに名前をつけるには，図13のように「Resource」メニューから「Get Resource Info」を選択します。図14のようなウィンドウが表示されるので「Name:」の部分に名前を入力してウィンドウを閉じます。

ちなみに，ここでリソース番号も変更することができます。リソース番号はリソースにより使用できる範囲が規定されています。多くの場合，128～32767までユーザーが使用できるようになっています。

開いているウィンドウを閉じていくと図15のようなウィンドウ画面になるはずです。PICTリソースが作成されていることを示すアイコンが表示されています。PICTリソースだけでなくサウンドやメニュー，アイコン，アプリケーション情報などなんでも作成可能です。68K Macではプログラムコードもリソースとなっています[*5]。

とりあえず，これで表示すべきPICT画像リソースは準備できました。

＊5　PowerPCはデータフォークにプログラムコードが格納されています。データフォークにPowerPCプログラム，リソースフォークに68Kプログラムが格納されているのがFAT Binaryと呼ばれるもので68KでもPowerPCでも動作するアプリケーションです。

PICTリソースを表示する

PICTリソースを表示させるにはPICTURE FIELDという命令を使います。この命令の書式は以下のようになっています。

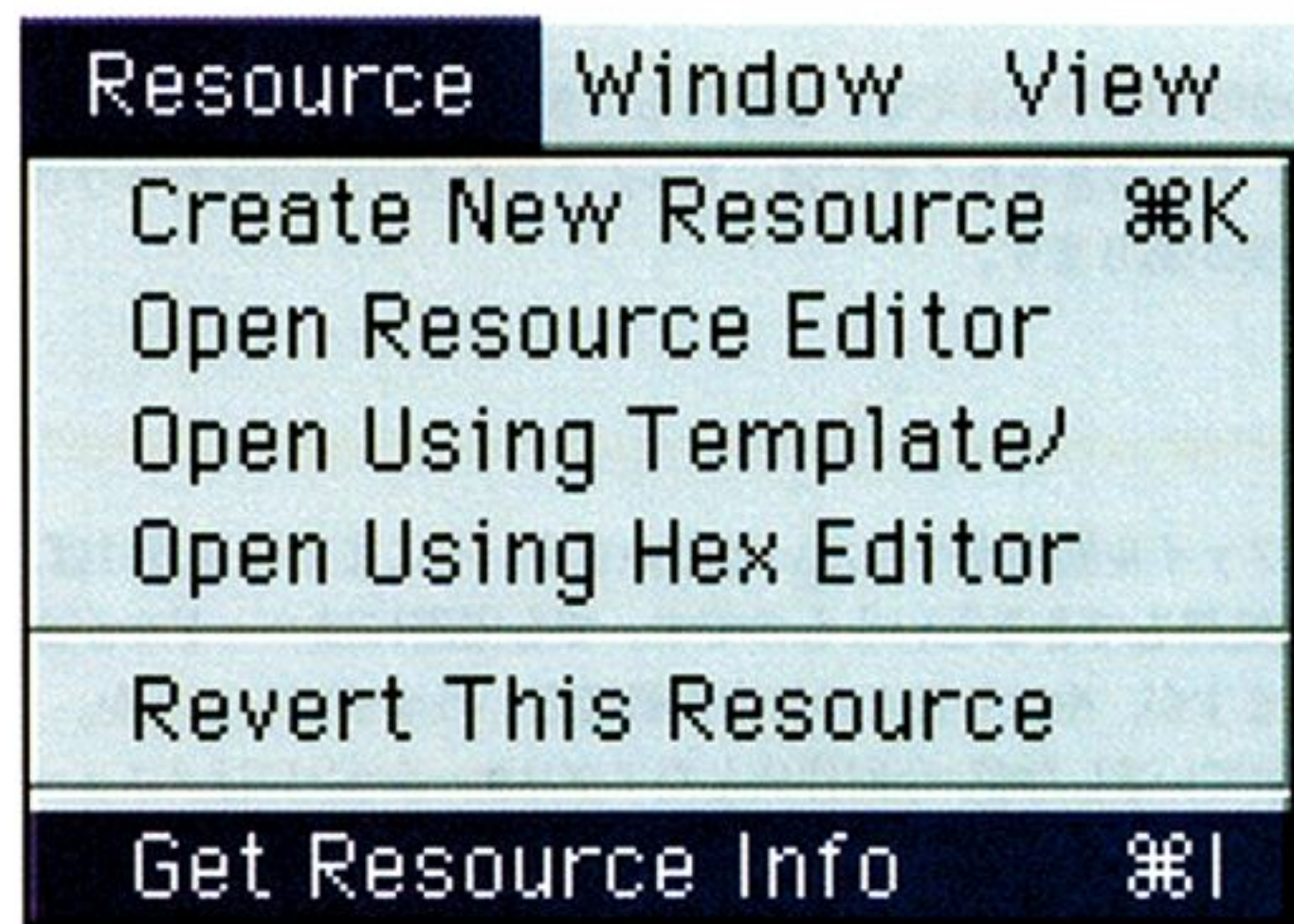



図13
各リソースの情報を設定変更する場合はResourceメニューからGet Resource Infoを選択

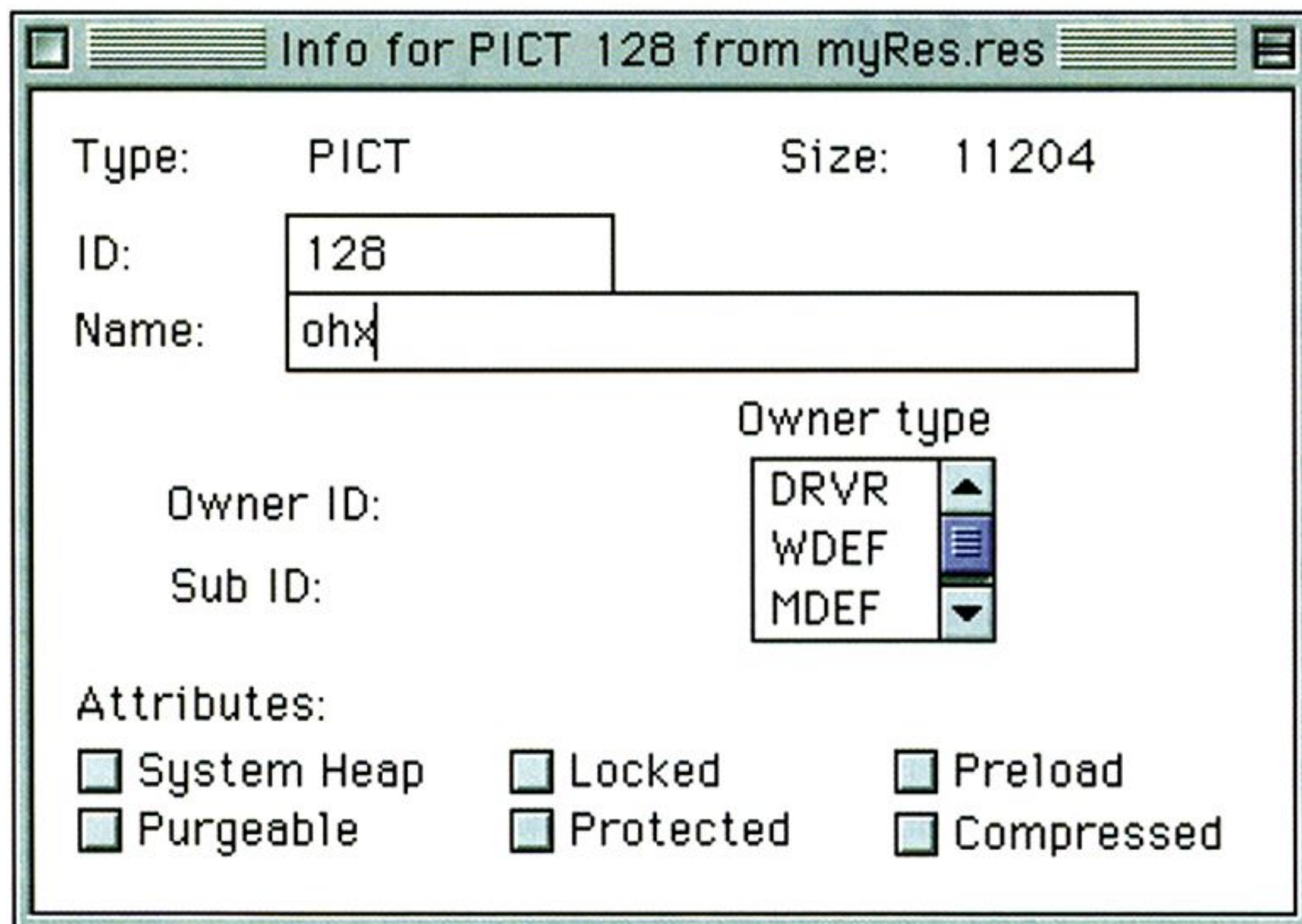



図14
IDや名前だけでなく属性も設定できる。ここではなにもチェック，指定しないこと

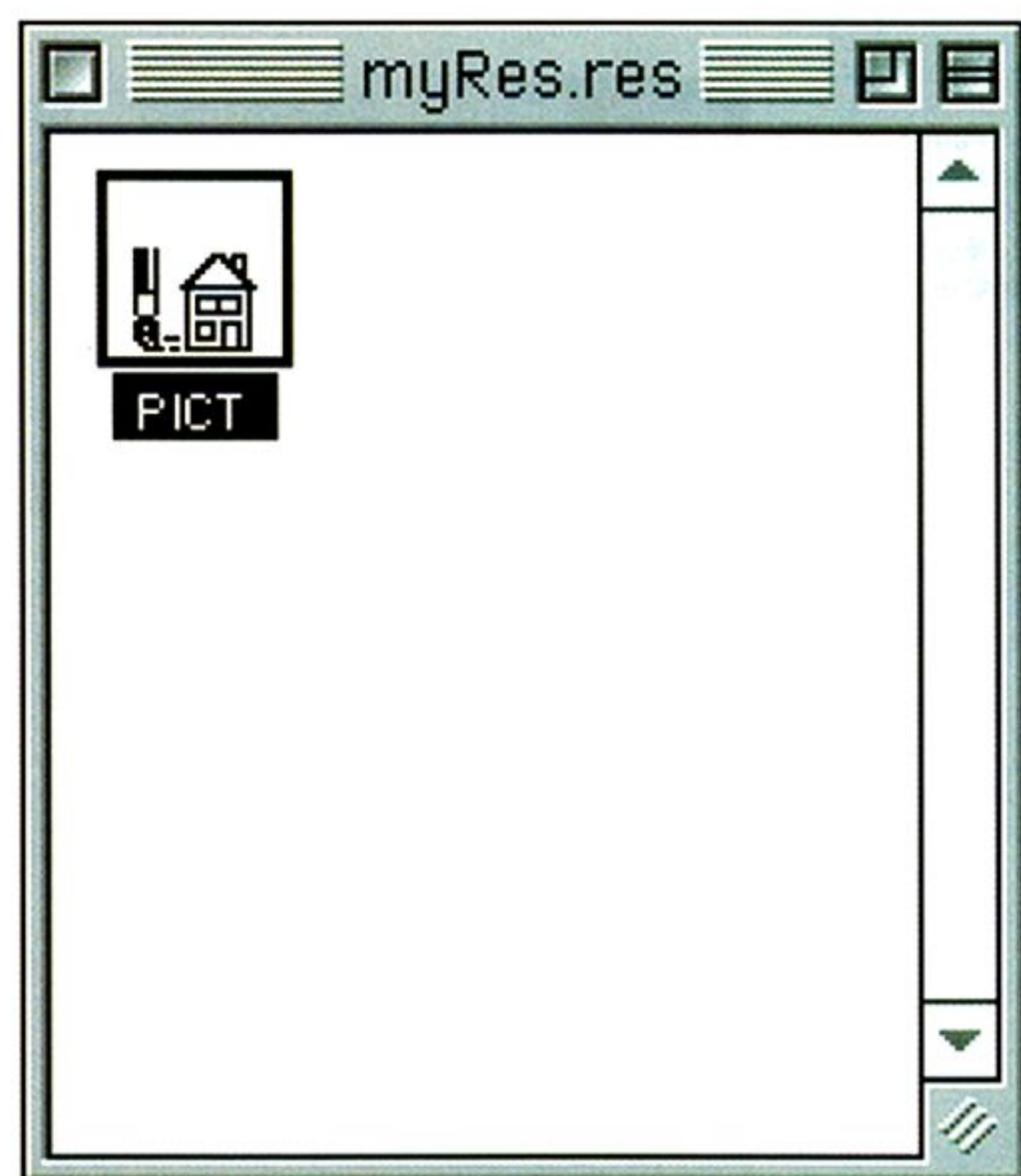



図15
PICTリソースがあることを示している

PICTURE FILED #フィールドID，%リソース番号，(x1，y1)-(x2，y2)，形式

フィールドIDというのはウィンドウ内でのPICTURE FILED固有の番号です。ほかのPICTURE FILEDで表示する画像およびEDIT FILED命令で使用されていない番号であれば大丈夫です。負数や浮動小数点数は使えません。1以上の整数値を使います。

リソース番号は先ほど作成したPICTリソースの番号です。さっきは128番だったので%128といった表記になります。PICTリソースに名前をつけてあるのであれば，PICTURE FILED #1，"ohx"といった具合にダブルクオーテーションで囲むことで表示させることが可能になります。

x1，y1，x2，y2はPICTリソースを表示する矩形サイズです。拡大縮小自由自在です。好みにあわせて表示させることができます。また，色数は表示しているモニタの色数に応じて処理され表示されますから，色数を気にすることはありません。もちろん綺麗に表示したいのであればフルカラーがベストです。

形式というのは表示する画像の周りに枠をつけるかどうかなどを指定するものです。表1に示すようにいくつもの種類があります。今回は枠なしですから_statNoFramedとなります。

PICTリソースを表示する命令はわかりましたが，これだけではPICTリソースは表示されません。Future BASICでリソースを扱う場合は明示的にリソースファイル名を指定し連結する必要があるのです。この連結する

表1

_framedNoCR	枠付きでクリック反転しない
_framed	枠付きでクリック反転あり
_noFramedNoCR	枠なしでクリック反転しない
_noFramed	枠なしでクリック反転あり
_statNoFramed	枠なしの固定画像
_statFramedGray	灰色の画像，枠付き固定
_statNoFramedGray	灰色の画像，枠なし固定
_statFramedInv	反転した画像，枠付き
_statNoFramedInv	枠なしの反転画像
_statKeepBg*	固定フィールドで背景を消さない
_noOutline*	アウトラインなし
_stat2Norm*	固定フィールドを標準にする
_stat2Gray*	固定フィールドを灰色にする
_stat2Inv*	固定フィールドを反転する
_hilite*	強調表示する
_round*	円形枠内に画像を表示
_rounder*	少し角が丸い四角形
_roundest*	角丸四角形
_boldBox*	強調枠

＊印のついているものは_statNoFramed_roundのように連続して記述し表示形式を指定する

命令がRESOURCESです。

RESOURCES "ファイル名"

のようにリソースファイル名を記述するだけで自動的に連結してくれます。

PICTリソースの128番を表示するのがプログラム1です。わずか7行です。PICTURE FILED命令の便利なところは手軽さだけでなく，アップデート処理[*6]（画面の自動書換処理）も行ってくれるところです。C/C++などではアップデート処理は独自に処理しないといけないのですが，Future BASICでは自動で行われるため単純に表示するだけで済んでしまうのです。

無事にPICT画像が表示できたので今度はアニメーションに挑戦してみましょう。

＊6　たとえばPICT画像を表示しているアプリケーションのウィンドウがほかのウィンドウに隠されたとします。次にPICT画像を表示しているアプリケーションに切り替わった場合，隠されていた部分を表示する必要がでてきます。これをアップデート処理と呼びます。通常アップデート処理は自前で実装するのですがFuture BASICでは勝手にFuture BASIC側で処理するためなにもする必要がないのです。

アニメーションさせる

アニメーションさせるには，アニメーションさせるだけのPICTリソース画像を用意します。ここでは面倒くさいので3Dアプリケーション（Infini-D ver 4.5J）でロゴを回転させアニメーションさせたものを使います。

PICTリソースの作成手順は一緒ですが，画像をコピーしてペーストする場合，自動的にID番号が加算されていきます。最初は128，次は129といった具合になります。

アニメーションさせる場合はPICTURE FIELD命令のリソース番号をカウントしていけばOKです。今回はID=128からID=167までの画像を繰り返し表示させることにします。PICTURE FILEDの%128の数値の部分を変数名，たとえば，Animにする（PICTURE FILED #1,%Anim）ことで次々と表示する画像を切り替えることができます。

プログラム2が画像をアニメーションさせるものです。特に難しい命令などは使用していませんが，繰り返し処理させるためにDO～UNTIL内で関数animeを呼び出しています。

実際にアプリケーション化して実行してみると，回転速度は一定しておらず，画面もやたらとちらついてしまいます。ちらつきはPICTURE FILEDを使う以上どうしようもありませんが回転速度は一定にすることが可能です。

回転速度を一定にするには前回時計を作成するときに使用したON TIMER命令を使います。時計は1秒間隔でしたが，今度は0.5秒，0.25秒など1秒以下の数値を指定しましょう。

ON TIMERで1秒以下の秒数を指定する場合，数値を負数とすることで1/60秒を基準とするようにできます。－60とすれば1秒，－30とすれば0.5秒になるわけです。1秒より短い時間で回転させたいので0.25秒，－15を指定します。

完成したものがプログラム3です。まだちらつきますが，これはどうしようもありません。ちらつきをなくすには，ちゃんとした表示処理を行うプログラムを組む必要があります。

おわりに

第3号では，ファイル処理をやりたいと思います。そこまで行き着けば，楽しい？　画像処理もできるというものです。でも実際にはハードルが高いような気がしますが，やはり日々の努力が必要というオチでしょうか。

おっと書き忘れていましたが，今回作成したアプリケーションにはメニューがありませんのでcommandキーとピリオドキーを押して終了させてください。

またFuture BASICのページも用意していますので参考にしてください。

URL http://www.shiojiri.ne.jp/~openspc/

リスト1　プログラム1

```
RESOURCES "myRes.res"
WINDOW OFF
WINDOW #1,"PICTリソース表示", (0,0) - (256,192) ,_dialogMovable
PICTURE FIELD #1,%128, (0,0) - (256,192) ,_statNoFramed

DO
  HANDLEEVENTS
UNTIL _false
```

リスト2　プログラム2

```
RESOURCES "myPict.res"
animeNo  = 128
animeMin = 128
animeMax = 167
END GLOBALS

' "アニメーションさせる関数"
LOCAL FN anime
  PICTURE FIELD #1,%animeNo, (0,0) - (256,192) ,_statNoFramed
  animeNo = animeNo + 1
  IF animeNo > animeMax THEN animeNo = animeMin
END FN

WINDOW OFF
WINDOW #1,"アニメーション", (0,0) - (256,192) ,_dialogMovable

' "メインイベントループ"
DO
  FN anime
  HANDLEEVENTS
UNTIL _false
```

図16
最後に保存する。あとFuture BASICを使う場合はResEditは必ず終了させること。Future BASICでアプリケーションが正常に作成されない場合がある。こうなってしまったら再起動させる以外方法がない

図17
復活Oh!Xの文字が回転する。速度はON TIMERの値を変更すれば調節可能

リスト3　プログラム3(完成)

```
RESOURCES "myPict.res"
animeNo  = 128
animeMin = 128
animeMax = 167
_animeW   = 256
_animeH   = 192
END GLOBALS

' "アニメーションさせる関数"
LOCAL FN anime
  PICTURE FIELD #1,%animeNo, (0,0) - (_animeW,_animeH) ,_statNoFramed
  animeNo = animeNo + 1
  IF animeNo > animeMax THEN animeNo = animeMin
END FN

WINDOW OFF
WINDOW #1,"アニメーション", (0,0) - (_animeW,_animeH) ,_dialogMovable

ON TIMER (-15)  FN anime

' "メインイベントループ"
DO
  HANDLEEVENTS
UNTIL _false
```

Macintosh PICT Format

●Macintosh PICT形式の概要

Macintoshの標準ファイル形式であるPICT形式について説明します。PICT形式には2つのバージョンがあります。バージョン1は初期のMacintoshで使用されていたもので1バイトのオプコードと続くデータによって定義されています。バージョン2はカラーQuickDrawとともに登場し現在まで使用されています。

バージョン2ではオペコードが2バイトに変更され，保存できる画像データも白黒からフルカラー画像まで可能になっています。PICT形式はビットマップ/ピクセルマップ画像だけでなく，テキストデータやライン，サークルなどのテキスト，ベクトルデータも格納することができます。またPicCommentという特殊なコードを利用することでEPSF（Encapsulated PostScript）形式も埋め込むことが可能です。QuickTimeがあればJPEG形式としても画像を保存することができます。

●PICT形式説明

PICT形式は先頭の512バイトがヘッダとなっていますが，ここは通常0でなにもありません。まれにアプリケーションによってデータが書き込まれることがありますが解析時には無視されます。

512～513バイト目には「最終データサイズ - 512」の演算結果の下位2バイトが格納されます。PICTバージョン1のときは画像データは32Kバイト以内という制限があり，そのときに画像データサイズを格納する場所でしたが，現在使用されているバージョン2では実質意味がありません。

次に続くのは画像サイズです。この画像サイズはPICTデータを72dpiで描いた場合のサイズです。たとえば144dpiの画像データがあり，100×50ピクセルであれば72dpiで描いた場合50×25ピクセル（72dpi/144dpi = 0.5 = 50%）となります。

そのため，ここで画像サイズを求め実際の解像度（つまりPICTデータに存在する）のピクセル数を決定することはできません。実際のピクセル数を求めるにはバージョン2固有のヘッダ内のデータを読み込む必要があります。Future BASICではPICT画像サイズは先頭にある72dpiで描いた場合のサイズを返します。画像処理ソフトなどを作成する場合は要注意です。

次にバージョンオプコードが続きます。ここはバージョン1と互換性を保つための，ちょっとした仕掛けがあります。ここのオペコードは以下のようになっています。

00 11 02 FF

バージョン1ではオプコードは1バイト単位で解釈されバージョン2では2バイト単位で解釈されます。この場合それぞれ以下のように解釈されます。

●バージョン1

00：なにもしない
11：以下の1バイトのバージョンを示すオペコード
02：バージョン2を示す
FF：終了を示すオプコード。ここで解析終了

●バージョン2

00 11：以下の2バイトのバージョンを示すオプコード
02 FF：バージョン2を示す。FFは無視される

これによりバージョン1しか解析できない場合，なにも処理されません。バージョン2の場合は次に続くオペコードが解析されていきます。

バージョン2では24バイトのヘッダが存在し画像のサイズ，解像度情報が格納されています。このヘッダは以下のようになっています。

2バイト：バージョン2ヘッダオペコード（0C00H）
2バイト：座標位置指定形式（－2）
2バイト：予約
4バイト：水平解像度
4バイト：垂直解像度
2バイト：左上Y座標
2バイト：左上X座標
2バイト：右下Y座標
2バイト：右下X座標
4バイト：予約（0）

水平解像度，垂直解像度の値は固定小数値となっており先頭2バイトが整数部分，後ろの2バイトが小数部となっています。

座標は水平解像度，垂直解像度での画像サイズです。実際のピクセル数を求める場合は，この座標値をもとに計算します。

バージョン2ヘッダの後ろに実際の描画オペコードが続きます。オペコードについては表にまとめておきましたので参考にしてください。バージョン1と2ではオペコードが異なります。

PICT画像を保存したアプリケーションによっては独自のPicCommentを埋め込んでいる場合があります。PicCommentは特殊な処理を行う場合を除き無視します。解析して処理する場合はPicCommentオペコードを認識したら続くKindコード（種類を示します）を読み込み，次に続く2バイトを読み込みます。この2バイトがPicCommentデータ長となっているので，この分だけスキップして解析すればよいことになります。

通常ヘッダの後ろには「クリッピング領域」が指定されています。オペコードに続く2バイト値がクリッピング座標値の数となっています。

以後，オペコードに対応したデータが続きます。

●画像データ

画像データを示すオペコードは，0090H～009FHです。実際には0090H，0091H，0098H，0099H，009AH，009BHが該当オペコードです。また8200H，8201Hオペコードに続くデータはQuickTimeによって解凍されるJPEGデータです。0090H，0091Hはバージョン1，2ともにビットマップ画像データを示すオペコードです。0090H，0091Hの場合は保存されているビットマップデータは圧縮されていません。0098H，0099Hの場合ビットマップデータはランレングス法（PackBits）によって圧縮されています。16ビット，32ビットカラーデータを示す009AHの場合はデータは圧縮されており同様のランレングス法で圧縮されます。0090H，0098Hで保存されるデータはマスク領域が矩形，0091H，0099Hはマスク領域が矩形以外となっています。（領域データによりマスク領域が設定されます）

画像データオペコードに続くデータはバージョン1と2で異なっており以下のようになっています。

・バージョン1

2バイト：ベースアドレス（未使用なので0）
2バイト：横方向の実際に必要なバイト数（rowBytes）
2バイト：左上Y座標
2バイト：左上X座標
2バイト：右下Y座標
2バイト：右下X座標
2バイト：元解像度での左上Y座標
2バイト：元解像度での左上X座標
2バイト：元解像度での右下Y座標
2バイト：元解像度での右下X座標
2バイト：72dpiでの左上Y座標
2バイト：72dpiでの左上X座標
2バイト：72dpiでの右下Y座標
2バイト：72dpiでの右下X座標
2バイト：転送モード

・バージョン2

2バイト：ベースアドレス（未使用なので0）

2バイト：横方向の実際に必要なバイト数(rowBytes)
2バイト：左上Y座標
2バイト：左上X座標
2バイト：右下Y座標
2バイト：右下X座標
2バイト：バージョン(常に0)
2バイト：圧縮タイプ(0)
4バイト：圧縮サイズ(0)
4バイト：水平解像度
4バイト：垂直解像度
2バイト：ピクセルタイプ
2バイト：1ピクセルあたりのビット数(4ビット=16色)
2バイト：次のピクセルまでのバイトオフセット(256色以下=1, 32768色=2, 1677万色=4)
2バイト：コンポーネントサイズ(1, 2, 4, 8, 16, 32ビット)
4バイト：次のカラープレーンまでのオフセット(0)
4バイト：反転(0)
4バイト：予約(0)
4バイト：カラーテーブル識別番号(存在しない場合は0)
2バイト：パレットフラグ(0)
2バイト：パレットデータ数
2バイト：パレット番号0(以下6バイトがパレット色データ)
6バイト：パレットデータ(RGB順)
　　：
　　：(パレットの数だけデータが繰り返される)
　　：
2バイト：元解像度での左上Y座標
2バイト：元解像度での左上X座標
2バイト：元解像度での右下Y座標
2バイト：元解像度での右下X座標
2バイト：72dpiでの左上Y座標
2バイト：72dpiでの左上X座標
2バイト：72dpiでの右下Y座標
2バイト：72dpiでの右下X座標
2バイト：転送モード

白黒，32768色，1677万色モード以外ではカラーパレットデータが存在します。パレットデータはパレット番号，赤，緑，青の順番にデータが並んでいますが，色を示す値は0～65535(0000H～FFFFH)までの2バイト(1ワード)です。1バイトでなく2バイトなのは描画を行うQuickDrawが扱う内部色情報が16ビットのためです。

表　転送モード一覧

名　称	値(10進数)	名　称	値(10進数)
srcCopy	0	notPatOr	13
srcOr	1	notPatXor	14
srcXor	2	notPatBic	15
srcBic	3	blend	32
notSrcCopy	4	addPin	33
notSrcOr	5	addOver	34
notSrcXor	6	subPin	35
notSrcBic	7	transparent	36
patCopy	8	adMax	37
patOr	9	subOver	38
patXor	10	adMin	39
patBic	11	grayishTextOr	49
notPatCopy	12	ditherCopy	64

転送モードは通常0です。しかし，ここに特定の値(2バイト)を設定することで通常以外の描画を行わせることが可能です。この転送モードはOh!X復刊1号77ページの「文字の表示モード」とほぼ同じです。設定できる値は表のとおりですが，描画モードによっては効果がないものがあります。もしAというアプリで作成されたPICT画像をBというアプリで描画させた場合，画像が汚くなるようであれば，PICTファイルのデータを変更して対処することも可能です。通常転送モードは0 = srcCopyになっているためディザがかかりません。これを64 = ditherCopy(16進数では40H)に変更すれば描画時に自動的にディザがかかります。こういう方法はすごくイレギュラーなもので普通は覚えていてもメリットはないかもしれません。

転送モードの後ろに実際の画像データが続きます。画像データは圧縮されている場合と圧縮されていない場合があります。0090H，0091Hのオプコードで示される画像データは圧縮されていません。また，rowBytesの値が8未満の場合でもデータは圧縮されていません。データが圧縮されていない場合は，データ＝ビットマップデータとなります。

圧縮データの前には圧縮データの長さを示す値(バイト値)があります。これに続いて以下の規則に従った圧縮データが続きます。

- 圧縮データは[長さ][データ....]の形式
- rowBytesが251以上であれば[長さ]は2バイト，251未満であれば[長さ]は1バイト

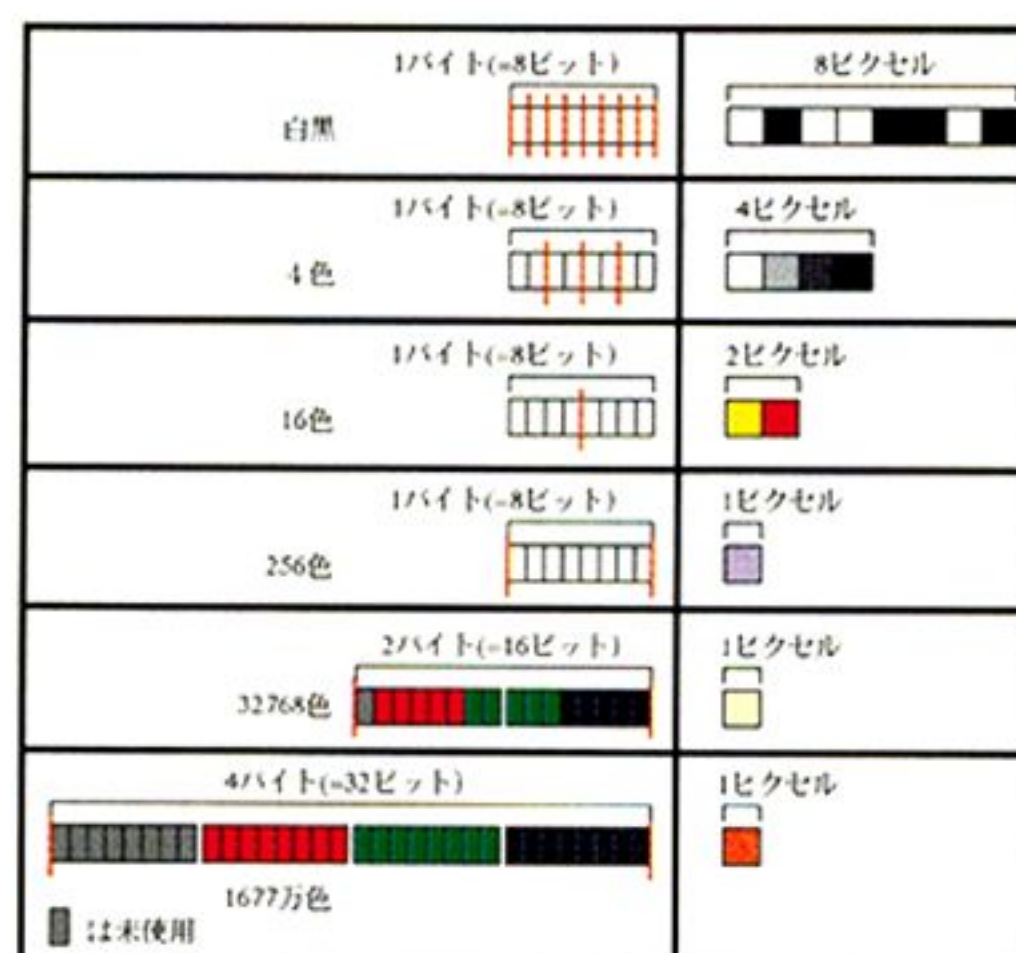
[長さ]が128以上か未満かにより続くデータの意味が異なります。[長さ]が128以上の場合は同じデータが指定数続き，128未満の場合は異なるデータが指定数続きます。指定された長さのデータは以下の計算式により求めます。

★128以上(連続ピクセル/ビットマップデータ)
　256－[長さ]＋1

★128未満(非連続ピクセル/ビットマップデータ)
　[長さ]＋1

データを展開し描画する場合は上記規則に従えばよいことになります。データは水平方向に圧縮され，垂直/ブロック圧縮などは行われません(8200H，8201HはJPEG圧縮)。

ピクセル/ビットマップの並び順ですが，これは図のようになっています。

図
色数とピクセルの構成

●最後に

MacintoshのPICT形式は結構面倒で解説した書籍もほとんどありません(InsideMacintosh，Graphic File Format)。

インターネット上では以下のアドレスでPICT形式についての説明を英語で読むことができます。

http://developer.apple.com/techpubs/mac/QuickDraw/QuickDraw-2.html

ちょっと実験，解析しつつやった部分もありますので，勘違いなどがあるかもしれません。間違っていたらごめんなさい。

以下に実際のPICTデータのダンプリスト/解析リストを示します。

24bit(RGB).pict

16×8サイズですべて白色。解像度は72dpi，保存形式は32ビットフルカラー。

```
00.......00 | 先頭512バイト(すべて0)
      00 A8 | データサイズ(168バイト)
      00 00 | 左上Y座標(0)
      00 00 | 左上X座標(0)
      00 05 | 右下Y座標(5)
      00 10 | 右下X座標(16)
      00 11 | バージョンオペコード(0011H)
      02 FF | バージョン番号(バージョン2なので02FFH)
      0C 00 | バージョン2ヘッダオペコード(0C00H)
      FF FE | 座標位置指定形式(-2)
      00 00 | 予約(0)
00 48 00 00 | 水平解像度(72dpi)
00 48 00 00 | 垂直解像度(72dpi)
      00 00 | 左上Y座標(0)
      00 00 | 左上X座標(0)
      00 05 | 右下Y座標(5)
      00 10 | 右下X座標(16)
00 00 00 00 | 予約(0)
      00 A1 | ロングコメント(00A1H)
      01 F2 | ロングコメント番号(01F2H)*0
      00 16 | ロングデータ長(22バイト)
38 42 49 4D 00 00 00 00 00 00 00 05 00 10 47 72 89 70 68 AF 62 6A |
ロングコメントデータ
      00 01 | クリップ領域オペコード(0001H)
      00 0A | クリップ領域：データサイズ+2(10バイト)
00 00 00 00 00 05 00 10 | クリップ領域：データ(ここでは矩形座標)
      00 9A | ピクセル圧縮オペコード(009AH)
00 00 00 FF | ピクセル圧縮データ：ベースアドレス(0：未使用)
      80 40 | ピクセル圧縮データ：横方向の実際に必要なバイト数，
                    rowBytes(16ピクセル×32ビット[4bytes]=64)
      00 00 | ピクセル圧縮データ：左上Y座標(0)
      00 00 | ピクセル圧縮データ：左上X座標(0)
      00 05 | ピクセル圧縮データ：右下Y座標(5)
      00 10 | ピクセル圧縮データ：右下X座標(16)
      00 00 | ピクセル圧縮データ：バージョン(これは常に0)
      00 04 | ピクセル圧縮データ：圧縮タイプ(4)
00 00 00 00 | ピクセル圧縮データ：圧縮サイズ(これは0)
00 48 00 00 | ピクセル圧縮データ：水平解像度(72dpi)
00 48 00 00 | ピクセル圧縮データ：垂直解像度(72dpi)
      00 10 | ピクセル圧縮データ：ピクセルタイプ
      00 20 | ピクセル圧縮データ：1ピクセルあたりのビット数(32ビット)
      00 03 | ピクセル圧縮データ：次のピクセルまでのバイトオフセット(RGBなので3)
      00 08 | ピクセル圧縮データ：コンポーネントサイズ(8ビット)
00 00 00 00 | ピクセル圧縮データ：次のカラープレーンまでのオフセット(0)
00 00 00 00 | ピクセル圧縮データ：反転(0)
00 00 00 00 | ピクセル圧縮データ：予約(0)
      00 00 | ピクセル圧縮データ：元解像度での左上Y座標(0)
      00 00 | ピクセル圧縮データ：元解像度での左上X座標(0)
      00 05 | ピクセル圧縮データ：元解像度での右下Y座標(5)
      00 10 | ピクセル圧縮データ：元解像度での右下X座標(16)
      00 00 | ピクセル圧縮データ：72dpiでの左上Y座標(0)
      00 00 | ピクセル圧縮データ：72dpiでの左上X座標(0)
      00 05 | ピクセル圧縮データ：72dpiでの右下Y座標(5)
      00 10 | ピクセル圧縮データ：72dpiでの右下X座標(16)
      00 40 | ピクセル圧縮データ：横一列のバイト数(16ピクセル×32ビット[4bytes])
   02 D1 FF
   02 D1 FF
   02 D1 FF
   02 D1 FF
   02 D1 FF | 圧縮されたデータ*1
         00 | Padding?
      00 FF | エンドコード
```

*0　01F2HはAdobe Photoshopの独自PicCommentのようです。続くデータは8BIM……となっていますので，クリエータなどのファイル情報などが格納されているようです。

*1
02：パックデータバイト長(2)
D1：データ長(同じデータの場合80H以上。FFH-D1H+1=48(描画リピート回数)。
　1ライン=16バイト×3ピクセル)
FF：実際の色の値(白なのでR=FFH,G=FFH,B=FFH)
　5ライン(5行)あるので，5つ同じものが繰り返されています。

24bit(RGB).pictダンプリスト

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0200 00 A8 00 00 00 00 00 05 00 10 00 11 02 FF 0C 00
0210 FF FE 00 00 00 48 00 00 00 48 00 00 00 00 00 00
0220 00 05 00 10 00 00 00 00 00 A1 01 F2 00 16 38 42
0230 49 4D 00 00 00 00 00 00 00 05 00 10 47 72 89 70
0240 68 AF 62 6A 00 01 00 0A 00 00 00 00 00 05 00 10
0250 00 9A 00 00 00 FF 80 40 00 00 00 00 00 05 00 10
0260 00 00 00 04 00 00 00 00 00 48 00 00 00 48 00 00
0270 00 10 00 20 00 03 00 08 00 00 00 00 00 00 00 00
0280 00 00 00 00 00 00 00 00 00 05 00 10 00 00 00 00
0290 00 05 00 10 00 40 02 D1 FF 02 D1 FF 02 D1 FF 02
02A0 D1 FF 02 D1 FF 00 00 FF
```

8bit(Index).pict

33×10サイズで上から「黒赤緑青白」の5色。解像度は144dpi。
保存形式は8ビットインデックスカラー(16色)。

```
00.......00 | 先頭512バイト(すべて0)
      01 02 | データサイズ(258バイト)
      00 00 | 左上Y座標(0)
      00 00 | 左上X座標(0)
      00 05 | 右下Y座標(5)
      00 11 | 右下X座標(17),72dpi描画時のサイズ
      00 11 | バージョンオペコード(0011H)
      02 FF | バージョン番号(バージョン2なので02FFH)
      0C 00 | バージョン2ヘッダオペコード(0C00H)
      FF FE | 座標位置指定形式(-2)
      00 00 | 予約(0)
00 90 00 00 | 水平解像度(144dpi)
00 90 00 00 | 垂直解像度(144dpi)
      00 00 | 左上Y座標(0)
      00 00 | 左上X座標(0)
      00 0A | 右下Y座標(10)
      00 21 | 右下X座標(33)
00 00 00 00 | 予約(0)
      00 A1 | ロングコメント(00A1H)
      01 F2 | ロングコメント番号(01F2H)
      00 16 | ロングデータ長(22バイト)
38 42 49 4D 00 00 00 00 00 00 00 05 00 11 47 72 89 70 68 AF 62 6A |
ロングコメントデータ
      00 01 | クリップ領域オペコード(0001H)
      00 0A | クリップ領域：データサイズ+2(10バイト)
00 00 00 00 00 0A 00 21 | クリップ領域：データ(ここでは矩形座標)
      00 98 | ピクセル圧縮オペコード(0098H)
      80 14 | ピクセル圧縮データ：横方向の実際に必要なバイト数，rowBytes(33ピク
                    セル×4ビット[0.5bytes]=17-->20)*2
      00 00 | ピクセル圧縮データ：左上Y座標(0)
      00 00 | ピクセル圧縮データ：左上X座標(0)
      00 0A | ピクセル圧縮データ：右下Y座標(10)
```

```
      00 21 | ピクセル圧縮データ：右下X座標(33)
      00 00 | ピクセル圧縮データ：バージョン(これは常に0)
      00 00 | ピクセル圧縮データ：圧縮タイプ(0)
00 00 00 00 | ピクセル圧縮データ：圧縮サイズ(これは0)
00 90 00 00 | ピクセル圧縮データ：水平解像度(144dpi)
00 90 00 00 | ピクセル圧縮データ：垂直解像度(144dpi)
      00 00 | ピクセル圧縮データ：ピクセルタイプ
      00 04 | ピクセル圧縮データ：1ピクセルあたりのビット数(4ビット=16色)
      00 01 | ピクセル圧縮データ：次のピクセルまでのバイトオフセット
                                 (16色インデックスカラーなので1)
      00 04 | ピクセル圧縮データ：コンポーネントサイズ(4ビット)
00 00 00 00 | ピクセル圧縮データ：次のカラープレーンまでのオフセット(0)
00 00 00 00 | ピクセル圧縮データ：反転(0)
00 00 00 00 | ピクセル圧縮データ：カラーテーブル識別番号(0)
00 43 65 5B | カラーテーブルID
      00 00 | カラーテーブルフラグ(0)
      00 04 | 登録されているパレット数(0から4。合計5)
      00 00 | パレット番号0(以下6バイトがパレット色データ)
FF FF FF FF FF FF | ピクセル圧縮データ：0番のパレット
                                 (R=FFFFH,G=FFFFH,B=FFFFH/白)
      00 01 | パレット番号1(以下6バイトがパレット色データ)
FF FF 00 00 00 00 | ピクセル圧縮データ：1番のパレット
                                 (R=FFFFH,G=0000H,B=0000H/赤)
      00 02 | パレット番号2 (以下6バイトがパレット色データ)
00 00 FF FF 00 00 | ピクセル圧縮データ：2番のパレット
                                 (R=0000H,G=FFFFH,B=0000H/緑)
      00 03 | ピクセル圧縮データ：パレット番号3(以下6バイトがパレット色データ)
00 00 00 00 FF FF | ピクセル圧縮データ：3番のパレット
                                 (R=0000H,G=0000H,B=FFFFH/青)
      00 04 | ピクセル圧縮データ：パレット番号4(以下6バイトがパレット色データ)
00 00 00 00 00 00 | 4番のパレット(R=0000H,G=FFFFH,B=0000H/黒)
      00 00 | ピクセル圧縮データ：元解像度での左上Y座標(0)
      00 00 | ピクセル圧縮データ：元解像度での左上X座標(0)
      00 0A | ピクセル圧縮データ：元解像度での右下Y座標(10)
      00 21 | ピクセル圧縮データ：元解像度での右下X座標(33)
      00 00 | ピクセル圧縮データ：72dpiでの左上Y座標(0)
      00 00 | ピクセル圧縮データ：72dpiでの左上X座標(0)
      00 0A | ピクセル圧縮データ：72dpiでの右下Y座標(10)
      00 21 | ピクセル圧縮データ：72dpiでの右下X座標(33)
      00 00 | ピクセル圧縮データ：転送モード
06 F1 44 00 40 FE 00
06 F1 44 00 40 FE 00
06 F1 11 00 10 FE 00
06 F1 11 00 10 FE 00
06 F1 22 00 20 FE 00
06 F1 22 00 20 FE 00
06 F1 33 00 30 FE 00
06 F1 33 00 30 FE 00
   02 ED 00
   02 ED 00 | 圧縮されたデータ*3
      00 FF | エンドコード
```

＊2　rowBytesの上位3ビットはフラグとして使用されている。そのため実際のrowBytes，つまり横幅を求めるには，

```
rowBytes = rowBytes AND &H03FF(Future BASIC)
rowBytes = rowBytes & 0x3FFF(C/C++)
```

のようにする必要がある。またrowBytesは16の倍数でなければならないため実際の画像の横幅よりも大きい値になる場合がある。

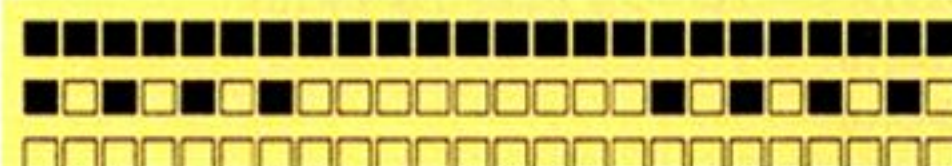
＊3

06：パックデータバイト長(6)
F1：データ長(同じデータの場合80H以上。100H-F1H+1=16。(描画リピート回数))
44：1バイトで2ピクセル分。16色なので1バイトに2ピクセル分のデータが入る。繰り返し数が16なので16×2=32。4はパレット番号
00：データ長(異なるデータの場合80H未満。00H+1=1。(描画リピート回数))
40：1ピクセルのみ。パレット番号は4。横が33ピクセルなので，その端数
FE：データ長(同じデータの場合80H以上。100H-FEH+1=3。(描画リピート回数))
00：パレット0のデータ。ピクセル幅を偶数にするために3ピクセル追加。パディング処理をする

8bit(Index).pictダンプリスト

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0200 01 02 00 00 00 00 00 05 00 11 00 11 02 FF 0C 00
0210 FF FE 00 00 00 90 00 00 00 90 00 00 00 00 00 00
0220 00 0A 00 21 00 00 00 00 00 A1 01 F2 00 16 38 42
0230 49 4D 00 00 00 00 00 00 00 05 00 11 47 72 89 70
0240 68 AF 62 6A 00 01 00 0A 00 00 00 00 00 0A 00 21
0250 00 98 80 14 00 00 00 00 00 0A 00 21 00 00 00 00
0260 00 00 00 00 00 90 00 00 00 90 00 00 00 00 00 04
0270 00 01 00 04 00 00 00 00 00 00 00 00 00 00 00 00
0280 00 43 65 5B 00 00 00 04 00 00 FF FF FF FF FF FF
0290 00 01 FF FF 00 00 00 00 00 02 00 00 FF FF 00 00
02A0 00 03 00 00 00 00 FF FF 00 04 00 00 00 00 00 00
02B0 00 00 00 00 00 0A 00 21 00 00 00 00 00 0A 00 21
02C0 00 00 06 F1 44 00 40 FE 00 06 F1 44 00 40 FE 00
02D0 06 F1 11 00 10 FE 00 06 F1 11 00 10 FE 00 06 F1
02E0 22 00 20 FE 00 06 F1 22 00 20 FE 00 06 F1 33 00
02F0 30 FE 00 06 F1 33 00 30 FE 00 02 ED 00 02 ED 00
0300 00 FF
```

1bit(BW).pict

24×3サイズで以下の画像。解像度は72dpi。保存形式は白黒。PICTバージョンは1。

```
■■■■■■■■■■■■■■■■■■■■■■■■
■□■□■□■□□□□□□□□□■□■□■□■□
□□□□□□□□□□□□□□□□□□□□□□□□

00.......00 | 先頭512バイト(すべて0)
      00 5C | データサイズ(92バイト)
      00 00 | 左上Y座標(0)
      00 00 | 左上X座標(0)
      00 03 | 右下Y座標(3)
      00 18 | 右下X座標(24),72dpi描画時のサイズ
         11 | バージョンオペコード (11H)
         01 | バージョン番号(バージョン1なので01H)
         A1 | ロングコメント(A1H)
      01 F2 | ロングコメント番号(01F2H)
      00 16 | ロングデータ長(22バイト)
38 42 49 4D 00 00 00 00 00 00 00 03 00 18 47 72 89 70 68 AF 62 6A |
ロングコメントデータ
         01 | クリップ領域オペコード(0001H)
      00 0A | クリップ領域：データサイズ+2(10バイト)
00 00 00 00 00 03 00 18 | クリップ領域：データ(ここでは矩形座標)
         90 | ビットマップ非圧縮オプコード(90H)
      00 04 | ビットマップ非圧縮データ：横方向の実際に必要なバイト数，rowBytes
                                 (24ドット/8ビット[1bytes]=3-->4)*4
      00 00 | ビットマップ非圧縮データ：左上Y座標(0)
      00 00 | ビットマップ非圧縮データ：左上X座標(0)
      00 03 | ビットマップ非圧縮データ：右下Y座標(10)
      00 18 | ビットマップ非圧縮データ：右下X座標(33)
      00 00 | ビットマップ非圧縮データ：元解像度での左上Y座標(0)
      00 00 | ビットマップ非圧縮データ：元解像度での左上X座標(0)
      00 03 | ビットマップ非圧縮データ：元解像度での右下Y座標(10)
      00 18 | ビットマップ非圧縮データ：元解像度での右下X座標(33)
      00 00 | ビットマップ非圧縮データ：72dpiでの左上Y座標(0)
      00 00 | ビットマップ非圧縮データ：72dpiでの左上X座標(0)
```

```
      00 03 | ビットマップ非圧縮データ：72dpiでの右下Y座標(10)
      00 18 | ビットマップ非圧縮データ：72dpiでの右下X座標(33)
      00 00 | ビットマップ非圧縮データ：転送モード
FF FF FF 00
AA 00 AA 00
00 00 00 00 | 圧縮されていないデータ*5
         FF | エンドコード
```

＊4　rowBytesは偶数で終わらせる必要があるため3バイトでなく4バイトとなります(パディング)

＊5　圧縮の効果がない場合，非圧縮データつまり生データとして保存されます。

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0200 00 5C 00 00 00 00 00 03 00 18 11 01 A1 01 F2 00
0210 16 38 42 49 4D 00 00 00 00 00 00 00 03 00 18 47
0220 72 89 70 68 AF 62 6A 01 00 0A 00 00 00 00 00 03
0230 00 18 90 00 04 00 00 00 00 00 03 00 18 00 00 00
0240 00 00 03 00 18 00 00 00 00 00 03 00 18 00 00 FF
0250 FF FF 00 AA 00 AA 00 00 00 00 00 FF
```

バージョン1

バイト数	内　容
512	ヘッダ(基本的にはゼロ)
2	データサイズ(ヘッダの512バイトは含まない)
2	画像の左上のY座標(Top)
2	画像の左上のX座標(Left)
2	画像の右下のY座標(Bottom)
2	画像の右下のX座標(Right)
1	バージョンオペコード(11H)
1	バージョン番号(01H)

以下指定オペコード

バージョン2

バイト数	内　容
512	ヘッダ(基本的にはゼロ)
2	データサイズ(ヘッダの512バイトは含まない。データサイズの下位2バイトが入る)
2	画像の左上のY座標(Top)
2	画像の左上のX座標(Left)
2	画像の右下のY座標(Bottom)
2	画像の右下のX座標(Right)
2	バージョンオペコード(0011H)
2	バージョン番号(02FFH)
2	バージョン2オペコード(00C0H)
2	形式(下記参照)

形式(FFFEHの場合)

2	予約(ゼロ)
4	水平解像度
4	垂直解像度
8	72dpiのときの矩形サイズ(Top,Left,Bottom,Right)
4	予約(ゼロ)

形式(FFFFHの場合)

2	FFFFH
4	固定小数点での画像の左上のY座標　(Top)
4	固定小数点での画像の左上のX座標　(Left)
4	固定小数点での画像の右下のY座標　(Bottom)
4	固定小数点での画像の右下のX座標　(Right)
4	予約　(ゼロ)

以下指定オペコード

バージョン1　オペコード

オペコード	データサイズ	データタイプ	内　容
00	0		なにも処理しない
01	領域データ	region	クリップ領域
02	8	pattern	背景パターン
03	2		書体ID
04	1		文字形状
05	2	mode	描画モード
06	4	fixedPoint	字間(固定小数点で指定)
07	4	point	ペンサイズ
08	2	mode	ペンモード
09	8	pattern	ペンパターン
0A	8	pattern	塗りつぶしパターン
0B	4	point	楕円サイズ
0C	4	dx,dy	描画開始点
0D	2	textsize	文字サイズ
0E	4		前景色
0F	4		背景色
10	8	point,point	文字縦横比
11	1		バージョン番号
12	0		予約
13	0		予約
14	0		予約
15	0		予約
16	0		予約
17	0		予約
18	0		予約
19	0		予約
1A	0		予約
1B	0		予約
1C	0		予約
1D	0		予約
1E	0		予約
1F	0		予約
20	8	x,y,x,y	線(開始点，終了点)
21	4	x,y	線(終了点)
22	6	point,dx,dy	線(ペン位置，相対位置)
23	4	dx,dy	線(相対位置)
24	0		予約
25	0		予約
26	0		予約
27	0		予約
28	5+文字列長	point,count,str255	文字
29	2+文字列長	dx,count,str255	文字(水平相対座標)
2A	2+文字列長	dy,count,str255	文字(垂直相対座標)
2B	3+文字列長	dx,dy,count,str255	文字(水平垂直相対座標)
2C	0		予約
2D	0		予約
2E	0		予約
2F	0		予約
30	8	rect	矩形描画(枠)
31	8	rect	矩形描画(ペイント)
32	8	rect	矩形描画(消去)
33	8	rect	矩形描画(反転)
34	8	rect	矩形描画(フィル)
35	0		予約
36	0		予約
37	0		予約
38	0	rect	最後に設定された矩形座標で描画(枠)
39	0	rect	最後に設定された矩形座標で描画(ペイント)
3A	0	rect	最後に設定された矩形座標で描画(消去)

オペコード	データサイズ	データタイプ	内容
3B	0	rect	最後に設定された矩形座標で描画(反転)
3C	0	rect	最後に設定された矩形座標で描画(フィル)
3D	0		予約
3E	0		予約
3F	0		予約
40	8	rect	角丸四角形描画(枠)
41	8	rect	角丸四角形描画(ペイント)
42	8	rect	角丸四角形描画(消去)
43	8	rect	角丸四角形描画(反転)
44	8	rect	角丸四角形描画(フィル)
45	0		予約
46	0		予約
47	0		予約
48	0	rect	最後に設定された矩形座標で角丸四角形描画(枠)
49	0	rect	最後に設定された矩形座標で角丸四角形描画(ペイント)
4A	0	rect	最後に設定された矩形座標で角丸四角形描画(消去)
4B	0	rect	最後に設定された矩形座標で角丸四角形描画(反転)
4C	0	rect	最後に設定された矩形座標で角丸四角形描画(フィル)
4D	0		予約
4E	0		予約
4F	0		予約
50	8	rect	楕円描画(枠)
51	8	rect	楕円描画(ペイント)
52	8	rect	楕円描画(消去)
53	8	rect	楕円描画(反転)
54	8	rect	楕円描画(フィル)
55	0		予約
56	0		予約
57	0		予約
58	0	rect	最後に設定された楕円の座標で再度楕円描画(枠)
59	0	rect	最後に設定された楕円の座標で再度楕円描画(ペイント)
5A	0	rect	最後に設定された楕円の座標で再度楕円描画(消去)
5B	0	rect	最後に設定された楕円の座標で再度楕円描画(反転)
5C	0	rect	最後に設定された楕円の座標で再度楕円描画(フィル)
5D	0		予約
5E	0		予約
5F	0		予約
60	12	rect,angle,angle	円弧描画(枠)
61	12	rect,angle,angle	円弧描画(ペイント)
62	12	rect,angle,angle	円弧描画(消去)
63	12	rect,angle,angle	円弧描画(反転)
64	12	rect,angle,angle	円弧描画(フィル)
65	0		予約
66	0		予約
67	0		予約
68	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(枠)
69	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(ペイント)
6A	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(消去)
6B	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(反転)
6C	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(フィル)
6D	0		予約
6E	0		予約
6F	0		予約
70	データサイズ	polygon	多角形描画(枠)
71	データサイズ	polygon	多角形描画(ペイント)
72	データサイズ	polygon	多角形描画(消去)
73	データサイズ	polygon	多角形描画(反転)
74	データサイズ	polygon	多角形描画(フィル)
75	0		予約
76	0		予約
77	0		予約
78	0		未実装。多角形描画(枠)
79	0		未実装。多角形描画(ペイント)
7A	0		未実装。多角形描画(消去)
7B	0		未実装。多角形描画(反転)
7C	0		未実装。多角形描画(フィル)
7D	0		予約
7E	0		予約
7F	0		予約
80	データサイズ	region	領域描画(枠)
81	データサイズ	region	領域描画(ペイント)
82	データサイズ	region	領域描画(消去)
83	データサイズ	region	領域描画(反転)
84	データサイズ	region	領域描画(フィル)
85	0		予約
86	0		予約
87	0		予約
88	データサイズ		未実装。領域描画(枠)
89	データサイズ		未実装。領域描画(ペイント)
8A	データサイズ		未実装。領域描画(消去)
8B	データサイズ		未実装。領域描画(反転)
8C	データサイズ		未実装。領域描画(フィル)
8D	0		予約
8E	0		予約
8F	0		予約
90	データサイズ		非圧縮ビットマップデータ(矩形)
91	データサイズ		非圧縮ビットマップデータ(領域)
92	0		予約
93	0		予約
94	0		予約
95	0		予約
96	0		予約
97	0		予約
98	データサイズ		圧縮ビットマップデータ(矩形)
99	データサイズ		圧縮ビットマップデータ(領域)
9A	0		予約
9B	0		予約
9C	0		予約
9D	0		予約
9E	0		予約
9F	0		予約
A0	2	kind	ショートコメント
A1	4+データ長	kind,length,data	ロングコメント
A2～FE	0		予約
FF	0		エンドコード

バージョン2　オペコード

オペコード	データサイズ	データタイプ	内容
0000	0		なにも処理しない
0001	領域サイズ	region	クリップ領域
0002	8	pattern	背景パターン
0003	2		書体ID
0004	1		文字形状
0005	2	mode	描画モード
0006	4	fixedPoint	字間(固定小数点で指定)
0007	4	point	ペンサイズ
0008	2	mode	ペンモード
0009	8	pattern	ペンパターン
000A	8	pattern	塗りつぶしパターン
000B	4	point	楕円サイズ
000C	4	dx,dy	描画開始点
000D	2	textsize	文字サイズ
000E	4		前景色
000F	4		背景色
0010	8	point,point	文字縦横比
0011	1		バージョン番号
0012	データサイズ	colorPattern	背景パターン(カラー)
0013	データサイズ	colorPattern	ペンパターン(カラー)
0014	データサイズ	colorPattern	塗りつぶしパターン(カラー)
0015	2		ペン位置
0016	2		文字間隔(カーニング)
0017	0		予約
0018	0		予約
0019	0		予約
001A	6	RGBcolor	前景色(RGB)
001B	6	RGBcolor	背景色(RGB)
001C	0		ハイライトモード
001D	6	RGBcolor	ハイライトカラー(RGB)
001E	0		デフォルトのハイライトカラー使用
001F	6	RGBcolor	RGB演算モード
0020	8	x,y,x,y	線(開始点, 終了点)
0021	4	x,y	線(終了点)
0022	6	point,dx,dy	線(ペン位置, 相対位置)
0023	4	dx,dy	線(相対位置)
0024	2+データ長		予約
0025	2+データ長		予約
0026	2+データ長		予約
0027	2+データ長		予約
0028	5+文字列長	point,count,str255	文字
0029	2+文字列長	dx,count,str255	文字(水平相対座標)
002A	2+文字列長	dy,count,str255	文字(垂直相対座標)
002B	3+文字列長	dx,dy,count,str255	文字(水平垂直相対座標)
002C	2+データ長		予約
002D	2+データ長	count,fixPoint,fixPoint	行間調整(Script Managerが使用)
002E	2+データ長		予約
002F	2+データ長		予約

オペコード	データサイズ	データタイプ	内容
0030	8	rect	矩形描画(枠)
0031	8	rect	矩形描画(ペイント)
0032	8	rect	矩形描画(消去)
0033	8	rect	矩形描画(反転)
0034	8	rect	矩形描画(フィル)
0035	8		予約
0036	8		予約
0037	8		予約
0038	0	rect	最後に設定された矩形座標で描画(枠)
0039	0	rect	最後に設定された矩形座標で描画(ペイント)
003A	0	rect	最後に設定された矩形座標で描画(消去)
003B	0	rect	最後に設定された矩形座標で描画(反転)
003C	0	rect	最後に設定された矩形座標で描画(フィル)
003D	0		予約
003E	0		予約
003F	0		予約
0040	8	rect	角丸四角形描画(枠)
0041	8	rect	角丸四角形描画(ペイント)
0042	8	rect	角丸四角形描画(消去)
0043	8	rect	角丸四角形描画(反転)
0044	8	rect	角丸四角形描画(フィル)
0045	8		予約
0046	8		予約
0047	8		予約
0048	0	rect	最後に設定された矩形座標で角丸四角形描画(枠)
0049	0	rect	最後に設定された矩形座標で角丸四角形描画(ペイント)
004A	0	rect	最後に設定された矩形座標で角丸四角形描画(消去)
004B	0	rect	最後に設定された矩形座標で角丸四角形描画(反転)
004C	0	rect	最後に設定された矩形座標で角丸四角形描画(フィル)
004D	0		予約
004E	0		予約
004F	0		予約
0050	8	rect	楕円描画(枠)
0051	8	rect	楕円描画(ペイント)
0052	8	rect	楕円描画(消去)
0053	8	rect	楕円描画(反転)
0054	8	rect	楕円描画(フィル)
0055	8		予約
0056	8		予約
0057	8		予約
0058	0	rect	最後に設定された楕円の座標で再度楕円描画(枠)
0059	0	rect	最後に設定された楕円の座標で再度楕円描画(ペイント)
005A	0	rect	最後に設定された楕円の座標で再度楕円描画(消去)
005B	0	rect	最後に設定された楕円の座標で再度楕円描画(反転)
005C	0	rect	最後に設定された楕円の座標で再度楕円描画(フィル)
005D	0		予約
005E	0		予約
005F	0		予約
0060	12	rect,angle,angle	円弧描画(枠)
0061	12	rect,angle,angle	円弧描画(ペイント)
0062	12	rect,angle,angle	円弧描画(消去)
0063	12	rect,angle,angle	円弧描画(反転)
0064	12	rect,angle,angle	円弧描画(フィル)
0065	12		予約
0066	12		予約
0067	12		予約
0068	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(枠)
0069	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(ペイント)
006A	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(消去)
006B	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(反転)
006C	4	rect,angle,angle	最後に設定された円弧の設定値で再度円弧描画(フィル)
006D	4		予約
006E	4		予約
006F	4		予約
0070	データサイズ	polygon	多角形描画(枠)
0071	データサイズ	polygon	多角形描画(ペイント)
0072	データサイズ	polygon	多角形描画(消去)
0073	データサイズ	polygon	多角形描画(反転)
0074	データサイズ	polygon	多角形描画(フィル)
0075	データサイズ		予約(ポリゴンデータ)
0076	データサイズ		予約(ポリゴンデータ)
0077	データサイズ		予約(ポリゴンデータ)
0078	0		未実装。多角形描画(枠)
0079	0		未実装。多角形描画(ペイント)
007A	0		未実装。多角形描画(消去)
007B	0		未実装。多角形描画(反転)
007C	0		未実装。多角形描画(フィル)
007D	0		予約
007E	0		予約
007F	0		予約
0080	データサイズ	region	領域描画(枠)
0081	データサイズ	region	領域描画(ペイント)
0082	データサイズ	region	領域描画(消去)
0083	データサイズ	region	領域描画(反転)
0084	データサイズ	region	領域描画(フィル)
0085	データ		予約(領域サイズ)
0086	データ		予約(領域サイズ)
0087	データ		予約(領域サイズ)
0088	データサイズ	region	未実装。領域描画(枠)
0089	データサイズ	region	未実装。領域描画(ペイント)
008A	データサイズ	region	未実装。領域描画(消去)
008B	データサイズ	region	未実装。領域描画(反転)
008C	データサイズ	region	未実装。領域描画(フィル)
008D	0		予約
008E	0		予約
008F	0		予約
0090	データサイズ		非圧縮ビットマップデータ(矩形)
0091	データサイズ		非圧縮ビットマップデータ(領域)
0092	2+データ		予約
0093	2+データ		予約
0094	2+データ		予約
0095	2+データ		予約
0096	2+データ		予約
0097	2+データ		予約
0098	データサイズ		圧縮ビットマップデータ(矩形)
0099	データサイズ		圧縮ビットマップデータ(領域)
009A	2+データ		予約
009B	2+データ		予約
009C	2+データ		予約
009D	2+データ		予約
009E	2+データ		予約
009F	2+データ		予約
00A0	2+データ	kind	ショートコメント
00A1	4+データ	kind,length,data	ロングコメント
00A2	2+データ		予約
00A3	2+データ		予約
00A4	2+データ		予約
00A5	2+データ		予約
00A6	2+データ		予約
00A7	2+データ		予約
00A8	2+データ		予約
00A9	2+データ		予約
00AA	2+データ		予約
00AB	2+データ		予約
00AC	2+データ		予約
00AD	2+データ		予約
00AE	2+データ		予約
00AF	2+データ		予約
00B0～00CF	0		予約
00D0～00FE	4+データ		予約
00FF	0		エンドコード
0100～01FF	2		予約
0200～02FE	4		予約
02FF	0		バージョン番号(02FFH)
0300～0BFF	6		予約
0400～0BFF	8		予約
0500～0BFF	10		予約
0600～0BFF	12		予約
0700～0BFF	14		予約
0800～0BFF	16		予約
0900～0BFF	18		予約
0A00～0BFF	20		予約
0B00～0BFF	22		予約
0C00	24		バージョン番号
0C01～7EFF	24		予約
7F00～7FFF	254		予約
8000～80FF	0		予約
8100～81FF	4+データ		予約
8200	4+データ		QuickTime圧縮データ
8201	4+データ		QuickTime非圧縮データ
8202～FFFF	4+データ		

★データタイプ

region：データ長(2バイト)＋領域座標データ(X, Y)

pattern：8×8ドットのパターン。横1バイト×縦8段＝8バイト。ビットマップデータ

mode：描画モードは以下の9つの種類が指定できる

srcCopy	0
srcOr	1
srcXor	2
srcBic	3
notSrcCopy	4
notSrcOr	5
notSrcXor	6
notSrcBic	7
grayishTextOr	49

fixedPoint：固定小数値。4バイトのうち先頭2バイトが整数部, 残りが小数部を表す

point：ポイント(1/72基準)

dx, dy：相対XY座標

x, y：XY座標

textsize：1～32767

count：文字列長を示す1バイト値(1～255)

str255：文字列(最大255文字)

rect：矩形座標を表すtop, left, bottom, rightのそれぞれ2バイト(合計8バイト)

angle：角度。負数も指定できる

polygon：データ長(2バイト)＋座標データ(X, Y)

kind：種類。2バイト

length：以下に続くデータ長。2バイト

RGBcolor：Red, Green, Blueそれぞれ2バイトの色情報

data：1バイト以上のデータ

colorPattern：カラーパターン。以下の形式
2バイト：パターンタイプ(1=2色パターン, 2=カラーパターン)

★パターンタイプが1の場合
8バイト：ビットマップパターン
6バイト：RGB(Red, Green, Blue)それぞれ2バイト(16ビット)

★パターンタイプが2の場合
8バイト：ビットマップパターン
2バイト：ベースアドレス(未使用なので0)
2バイト：横方向の実際に必要なバイト数(rowBytes)
2バイト：左上Y座標
2バイト：左上X座標
2バイト：右下Y座標
2バイト：右下X座標
2バイト：バージョン(常に0)
2バイト：圧縮タイプ(0)
4バイト：圧縮サイズ(0)
4バイト：水平解像度
4バイト：垂直解像度
2バイト：ピクセルタイプ
2バイト：1ピクセルあたりのビット数(4ビット=16色)
2バイト：次のピクセルまでのバイトオフセット
(256色以下=1, 32768色=2, 1677万色=4)
2バイト：コンポーネントサイズ(1, 2, 4, 8, 16, 32ビット)
4バイト：次のカラープレーンまでのオフセット(0)
4バイト：反転(0)
4バイト：予約(0)
4バイト：カラーテーブル識別番号(存在しない場合は0)
2バイト：パレットフラグ(0)
2バイト：パレットデータ数
2バイト：パレット番号0(以下6バイトがパレット色データ)
6バイト：パレットデータ(RGB順)
:
:(パレットの数だけデータが繰り返される)
:
nバイト：実際のピクセルデータ

PicComment番号

値	サイズ	内容
150	6	テキスト処理開始
151	0	テキスト処理終了
152	0	文字列処理開始(間隔処理)
153	0	文字列処理終了(間隔処理)
154	8	回転の中心座標オフセット
155	0	LaserWriterレイアウトオフ
156	0	LaserWriterレイアウトオン
157	16	カスタムラインレイアウト＊6
160	0	ベジエ曲線開始
161	0	ベジエ曲線終了
163	0	ポリゴンデータを無視
164	1	Fill, Frame終了
165	0	ポリゴン終了
180		破線描画
181	0	破線終了
182	4	(小数値の)線幅設定
190	0	PostScriptドライバ設定
191	0	QuickDrawの状態を元に戻す
192		PostScriptデータ(ハンドル)
193		ファイル名(ハンドル)＊7
194	0	QuickDrawテキストをPostScriptデータとして送信＊7
195	8	リソースフォークにあるPostScriptデータ＊7
196	0	PostScript命令を設定
197	4	PostScript命令のsetgrayを呼び出す(グレー値設定)＊6
200	4	開店開始
201	0	回転終了
202	8	回転の中心へのオフセット
210	0	プリンタバッファクリア＊6
211	0	プリンタ送信終了＊6

＊6　使用しないこと

＊7　無視される

Tcl/Tkお気楽GUIプログラミング 入門編

広井 誠　Hiroi Makoto

GUI環境のプログラムというと面倒なものですが，Tcl/Tkでは，シンプルなインタプリタ言語Tclと汎用GUI環境のTkが合体して，非常に簡単にGUIプログラムを構築できるようになっています。機種を問わずにGUIアプリが制作できるのも魅力のシステムです。

はじめに

Windowsが普及するようになって，パソコンを使う人は大幅に増えましたが，逆にプログラミングを楽しむ人は減ってきているようです。パソコンはソフトがなければただの箱，といわれますが，Windowsにはたくさんのアプリケーションやツールが市販されているので，目的にあったソフトを購入すれば，わざわざプログラムを作らなくてもパソコンを使うことができます。

ところが，優秀なアプリケーションでも長く使っているとなにかしらの不満が出てくるものです。一般のユーザーであればバージョンアップされるまで待つことになるでしょうが，ちょっとしたプログラムを作ることができれば，使い勝手を大幅に改善できる場合もあるのです。小さなツールであれば，個人でもプログラミングすることは十分に可能ですし，なによりも自分の好みにあわせて作ることができます。インターネットで見つけたツールが，いまいち手に馴染まないのであれば，あなたもプログラミングに挑戦してみましょう。

そうはいっても，プログラミングは難しい，と考えている人が多いと思います。特に，WindowsのようなGUI(Graphical User Interface)に対応したアプリケーションを作ることは，以前のCUI(Command User Interface)に比べて，相当の労力を必要とします。最新の開発環境を用意すれば，Windows の機能を簡単に使えるようになりますが，初心者にとって付属のマニュアルだけでは心細いので(理解できないともいう)，いろいろな参考書や資料が必要となります。つまり，開発環境を整えるにも，それなりの出費を覚悟しないといけません。

GUIの難しさに加え，懐具合も気にしなくてはならないようでは，プログラミングを楽しむどころではない，という方も多いでしょう。X68000のように，フリーで使用できる開発環境がWindowsにもあれば，それも手軽にGUIアプリケーションを作れるのであれば，プログラミングに挑戦してみよう，というユーザーもいるはずです。実は，そのような用途にぴったりの開発環境があるのです。それがTcl/Tkです。

Tcl/Tk(ティクルティケイ)は，UNIXのX Window，Mac OS，MS Windowsで，GUIアプリケーションを作成することができます。Tcl/Tkは，1988年にJohn K.Ousterhout博士によって開発されました。TclはTool Command Languageの略で，もともとはアプリケーションに組み込むためのコマンド言語として設計されたのですが，GUIの部品を集めたTk(Tool Kit)と組み合わせることで，Tcl/TkだけでGUIアプリケーションを簡単に作ることができるようになりました。わずか十数行のプログラムでも，十分に実用となるのですから驚きです。そして，なんといってもフリーで使用できるのが素晴らしいですね。それでは，簡単にTcl/Tkの世界を紹介していくことにしましょう。以下ではWindows版を中心に解説しますが，MacintoshでもX Windowでも基本は同じです。

Tcl/Tkとはどんな言語？

最初に説明したように，TkはGUIの部品を集めたものです。Tkを操作するためのプログラミング言語がTclです。Tclはインタプリタ方式の言語です。Windows版の標準の配布キットには，tclshとwishというアプリケーションが含まれています。tclshはTcl言語のみのインタプリタで，これにTkを加えたものがwish(WIndow SHell)です。GUIアプリケーションを実行する場合は，こちらを使います。wishを単独で起動すると，コンソールとwishのウィンドウが現れます。コンソールからプログラムを入力して実行することができるので，動作がよくわからない場合は手軽に試して確認することができます。

TclはC言語やBASICのような普通のプログラミング言語とはちょっと違います。その特徴をひと言でいえば，UNIXで使用されているシェル，bashやcshのシェルスクリプトを拡張した言語，といえるでしょう(MS-DOSでいえばバッチ処理に相当しますが，シェルスクリプトのほうがずっと高機能です)。

ところで，旧Oh! Xの読者であれば，シェルスクリプトを拡張したコマ

図1　たった2行の入力でボタンが表示された

図3　Windowsなら Help機能を活用しよう

図2　ボタンを3つ表示してみた

ンド言語という説明だけで，Tclに対する大まかなイメージをつかんでもらえるでしょうが，Windows全盛のこの時代，MS-DOSプロンプトなんて立ち上げたこともない，当然ながらバッチ処理はわからない，それ以前にコマンドってなに？　という読者もいると思いますので，簡単に解説しておきます。

コマンドはアプリケーションプログラムのことと考えてください。Windowsの場合，アプリケーションを実行するにはアイコンをダブルクリックすればいいですね。ところがCUIの場合，ユーザーはコンピュータに対する命令（コマンド）をキーボードから入力します。アプリケーションを実行する場合は，そのファイル名をキーボードから入力します。そして，ユーザーが入力した命令を処理するプログラムが「シェル」なのです。コマンドは次のような形式で入力します。

```
コマンド名 引数1 引数2 …… <RETURN>
```

最初はコマンド名で，その後ろに引数が必要になる場合もあります。引数にはコマンドのオプションを指定するフラグがあります。MS－DOSでは「／」の後ろの1文字で表しますが，UNIX系のOSでは「－」の後ろの1文字で表す場合が多いようです。このほかにも引数でファイル名やディレクトリ名，あるいは特別なオプションを指定する場合もあります。これは，コマンドによって異なります。

コマンド名や引数の間は空白で区切ります。そして最後にリターンキーを入力すると，シェルがこのコマンドを実行します。

Tcl でのプログラミングも，これが基本的な形式です。たとえば，ウィンドウにボタンを表示してみましょう。wishのコンソールから次のように入力してください。

```
% button .b －text "button 1"
.b
% pack .b
%
```

wishのウィンドウにボタンが現れましたね。実際にボタンを押してみてください。マウスの左ボタンを押すと，ウィンドウ上のボタンがへこみ，マウスの左ボタンを離すと，ウィンドウ上のボタンは元の形に戻ります。たった2行で，ここまでのことができるのです。

Tk ではGUIの部品をウィジェット（widget）と呼びます。この例では，ボタンウィジェットを使いましたが，このほかにもTkにはさまざまなウィジェットが用意されています。配布キットにはWidget Tourというウィジェットを紹介するのデモプログラムが用意されているので，ひととおり眺めてみてください。その機能の多さに驚かれることでしょう。

buttonはボタンを作るコマンドで，そのウィジェット名が.bとなります。－textはボタンに表示される名前を指定するオプションです。Tcl/Tkは出身がUNIXなので，オプションの指定は － から始まります。省略された場合は，デフォルトの値が使われます。実際に－textを省略すると，名前のないボタンが出現します。packはウィンドウにボタンを詰め込むコマンドです。いまはボタンを表示するだけですが，押したときの動作も指定することができます。

この例からもわかるように，Tclでは最初にコマンド名，その後ろに引数やオプションが与えられます。まさしく，シェルでのコマンド入力とそっくりですね。該当するコマンドがない場合はエラーとなります。作られたウィジェット名.bは，そのウィジェットを操作するためのコマンドとして働きます。これをウィジェットコマンドと呼びます。これはあとで詳しく説明します。

ところで，コンソールから複数のコマンドを入力するのは面倒ですね。複数のコマンドをまとめてひとつのコマンドとして実行するのがバッチ処理であり，シェルスクリプトなのです。MS-DOSの場合，拡張子batのファイルに書かれているコマンドは順番に実行されていきます。バッチファイルの中では，コマンド以外にも制御命令を使うことができます。DOSのバッチファイルで使用できる命令は貧弱なものですが，UNIXのシェルスクリプトでは一般のプログラミング言語に負けないくらいの命令が用意されています。

Tcl/Tkの場合は，拡張子がtclのファイルにプログラムを書きます。Tcl/Tkが正常にインストールされると，このファイルとwishが対応づけられます。あとは，このファイルをダブルクリックするとプログラムが実行されます。

たとえば，button0.tclに次のコマンドを書きましょう。

```
button .b1 －text "button 1"
button .b2 －text "button 2"
button .b3 －text "button 3"
pack .b1 .b2 .b3
```

このファイルをダブルクリックするとボタンが3つ表示されます。この場合，wishのコンソールは表示されません。このように，Tcl/Tkで作ったプログラムは，一般のアプリケーションと同じように実行することができます。

Tclの基礎知識

ここからは，Tclを中心に話を進めていきましょう。Tclの特徴は，データを文字列として扱うことです。文字列は空白かタブで区切られ，行の最初の文字列がコマンド名となります。該当するコマンドがない場合はエラーとなります。コマンドに渡される引数も文字列で渡されます。数値が必要な場合は，コマンド内部で文字列から数値に変換されます。このとき，数値に変換できない場合はエラーとなります。

●数値

数値データは，整数と実数（浮動小数点数）が用意されています。整数値の範囲は －2147483648から2147483647まで，つまり32ビット整数です。これは普通のC言語でのintと同じです。0から始まる整数は8進数とみなされ，0xから始まる整数は16進数とみなされます。実数値は，約－1e308から1e308まで，普通のC言語でのdoubleと同じ値です。表記法はC言語と同じで，1e308は1E308でも1E+308でも同じ値として認識されます。

●四則演算

Tcl では四則演算を行うにもコマンドが必要になります。exprは与えられた引数全体を式として評価し，その結果を文字列として返します。

```
% expr 1 + 2
```

```
3
% expr 3 * 4
12
```

exprの場合，数値と演算子の間は空白で区切らなくても正しく評価されます。使用できる演算子はC言語とほぼ同じで，＋－＊/％などの算術演算子のほかに，ビット演算子も用意されています。また，sin ()，cos () などの数学関数も使うことができます。詳細はexprのヘルプを参照してください。

データを格納する入れものを「変数」といいます。変数にデータを入れることを代入といい，Tcl/Tkではsetコマンドを使います。

```
% set a 2
2
% set a
2
```

変数aに2を代入しました。Tcl/Tkでは，英大文字と英小文字を区別します。aとAでは異なる変数になるので注意してください。変数を使う場合，あらかじめ宣言しておく必要はありません。setで変数に代入するデータを省略すると，その変数に格納されている値を返します。

●変数置換

変数から値を取り出すには，変数名の前に$をつけます。これを「変数置換」といいます。

```
% set b $a
2
% set c a
a
```

変数aの値をbに代入します。aの前に$がついているので，aに格納されている値2に置き換えられてからsetコマンドが実行され，bに2が代入されます。最後の例では，$がついていないので，aを文字列としてそのままcに代入します。

コマンドの評価結果を利用する場合は「コマンド置換」を使います。たとえば，exprの結果を変数に代入してみましょう。

```
% set a [expr 2 + 3]
5
% set b [expr $a * 4]
20
```

コマンド置換はブラケット（[]）で表します。ブラケットの中で最初の文字列がコマンドとなります。コマンド置換は，ブラケット内のコマンドを評価して，ブラケットをその結果に置き換えます。最初の例では，expr 2 + 3が評価されて，その結果である5に置き換えられます。それから setコマンドが評価され，変数に5が代入されます。変数置換はブラケット内でも有効です。次の例では，最初に変数$aの置換が行われ，それからexpr 5 ＊ 4が評価されます。そのあとでsetコマンドが評価され，変数bに20が代入されます。

●文字列の扱い

文字列に空白を含めたい場合は，文字列をダブルクォート（"）で囲みます。これはすでにボタンを表示するプログラムで使っています。ダブルクォートの内側でも，変数置換やコマンド置換が行われます。これを避けるにはエスケープ記号￥を使います。

```
% set c "a+b = [expr $a+$b]"
a + b = 25
% set d "a+b = ¥[expr ¥$a+¥$b]"
a + b = [expr $a+$b]
```

最初の例はコマンド置換が行われるので，$aと$bを足した値，25に置き換えられ，変数cには文字列"a + b = 25"が代入されます。ブラケット[と$の前に￥をつけると，置換が行われずにそのままの文字列が変数に代入されます。このほかにもエスケープ記号￥は，￥nや￥tで改行やタブを表したり，長いコマンドを複数行に分けて書く（継続行）ためにも使われます。

配列

配列は複数のデータを格納するデータ構造です。一般に，配列の要素は正の整数値を使って指定しますが，Tclでは文字列で要素を指定します。いわゆるPerlやawkで使われている連想配列のことで，C言語やBASICの配列とは違います。配列は名前の後ろに（）をつけ，その中にインデックス名を入れます。値を取り出すときは変数と同じく，名前の前に$をつけます。変数と同様に，あらかじめ配列を宣言しておく必要はありません。次の例を見てください。

```
% set table (1) 10
10
% set table (abc) $table (1)
10
% set x 1
1
% set y 2
2
% set table ($x,$y) 100
100
% set table (1,2)
100
```

table (1) に10を代入します。この場合でも1は数値ではなく文字列として扱われています。次の例では，table (1) の値をtable (abc) にセットします。

また，インデックス名に変数を使うこともできます。table ($x, $y) は変数置換が行われ，table (1, 2) に100が代入されます。この場合，インデックス名は1，2という文字列となります。この機能を使って多次元配列をシミュレートすることができます。これはPerlやawkでも使われるテクニックです。

リストと制御構造

リストは配列と同様に，複数のデータを格納することができますが，連想配列とは違い，要素を順番に並べただけのデータ構造です。リストは全体を{}で括り，要素を空白文字（空白，タブ，改行など）で区切ります。

```
% set a {10 20 30 40 50}
10 20 30 40 50
% set b {abc def ghi jkl}
abc def ghi jkl
% set c {abc {def ghi} jkl}
abd {def ghi} jkl
```

最後の例のように，リストの中にリストを入れることができます。ところで，リスト処理というとLispというプログラミング言語が有名ですね。Tclではリストを操作するコマンドが用意されていますが，Lispのようなプログラミングができるわけではありません。Tclではリストを頻繁に使いますが，それはifやwhileといった制御構造を実現するコマンドに，引数を渡すために用いられるからです。

● if

Tclの場合，リストの中では変数置換やコマンド置換は行われず，エスケープ記号￥も意味を失います。改行で要素を区切った場合も，改行コードがそのまま格納されます。簡単なifを使って説明しましょう。ifは条件分岐を行うコマンドです。次の例を見てください。

```
if {$i < 0} {set i 0}
```

ifは最初の引数を条件式として評価し，それが成立した場合は2番目の引数を評価します。関係演算子はC言語と同様に，＜（小さい），＜=（以下），＞（大きい），==（等しい），!=（等しくない）が用意されています。演算の結果は，条件を満たすならば1（真），そうでなければ0（偽）となります。関係演算子は文字列にも適用することができます。

引数はリストとして渡されていることに注意してください。もしリストを使わないと，第1引数が$i，第2引数が ＜ になってしまいます。set i 0もリストでなければいけません。そうしないと，ifの第2引数は set だけになってしまいエラーとなります。また，次のように書いてもエラーになります。

```
if {$i < 0}
{
    set i 0
}
```

この場合，ifの第2引数はなしと判断されエラーとなります。第2引数は，次のようにifと同じ行から書いてください。

```
if {$i < 0} {
    set i 0
}
```

複数のコマンドを実行したい場合は，コマンドを改行で区切りましょう。

```
if {$i < 0} {
    set i 0
    set j 1
    set k 2
}
```

Tclはシェルと同様に，基本的には1行を1コマンドとして認識します。リストの中でも，改行で区切られた要素までをコマンドとして認識します。もしも，1行に複数のコマンドを書きたい場合はセミコロン(;)で区切ります。

```
if {$i < 0} {
    set i 0; set j 1; set k 2
}
```

これはUNIX系のシェルと同じ使い方です。このほかに，ifはelse，elseifを使うことができます。

```
if {$i < 0} {
    set i 0
} elseif {$i < 10} {
    set i 10
} else {
    set i 100
}
```

elseifは何個でもつなげることができます。ただし，次のような書き方ではエラーになります。

```
if {$i < 0} {
    set i 0
}
else {
    set i 10
}
```

この場合，elseのないifと判断され，次のelseがコマンド名と解釈されるのです。このような書き方に慣れているユーザーには，ストレスが溜まるところでしょうが，Tclを使うときは我慢してください。

繰り返し

● while

次は，繰り返しを行うコマンドを説明します。最初はwhileです。

```
while 条件式 ループ本体
```

whileは第1引数を条件式として評価し，それが成立している間，第2引数のループ本体を繰り返し実行します。引数はリストで渡したほうが安全です。次の例を見てください。

```
while {$n > 0} {
    ……
    incr n -1
}
```

変数nに格納されている回数だけ処理を繰り返します。incrは第1引数の変数に第2引数の値を足すコマンドです。第2引数が省略されると+1されます。C言語の演算子+=や++に相当します。ところで，incrと逆の働きをするコマンドdecrもありそうなものですが，Tclでは用意されていないので注意してください。この処理を次のように書くと無限ループになります。

```
while $n {
    ……
    incr n -1
}
```

ほかの言語に慣れている方は，nが0になれば条件が不成立になるので繰り返しが終了する，と思うでしょうが，Tclの場合，引数は変数置換が行われてからコマンドに渡されるので，条件式はただの数値になってしまうのです。したがって，条件式の値は変化せず無限ループになるのです。ここは，変数置換が行われないリストに格納しておかないと正常に動作しないのです。Tclの場合，条件式は必ずリストで渡すと，覚えておいたほうがいいでしょう。

● for

次はforコマンドを説明します。

```
for 初期化式 条件式 更新式 ループ本体
```

for 自体の働きはC言語と同じですが，引数はリストで渡します。次の例を見てください。

```
for {set i 0}{$i < 10}{incr i}{
    ……
}
```

これをwhileで書き換えると次のようになります。

```
set i 0
while {$i < 10} {
    ……
    incr i
}
```

forはwhileに比べ，繰り返しを制御するコマンドをまとめて書ける，という長所があります。もちろん，どちらのコマンドを使うかはプログラマの自由で，わかりやすいほうを選べばよいのです。

●foreach

もうひとつ繰り返しを行うコマンドを説明します。UNIX系のシェルやPerlでお馴染みのforeachです。

```
foreach 変数 リスト ループ本体
```

foreachはリストから要素を順番に取り出して変数にセットし，ループ本体を実行します。次の例を見てください。

```
foreach n {0 1 2 3 4} {
    puts $n
}
```

putsは標準出力にデータを書き出すコマンドです。wishのコンソールで実行すると，0から4までの数値が出力されます。変数nにリストの要素が順番にセットされるのがわかるでしょう。これは次のように書き換えても正常に動作します。

```
set l {0 1 2 3 4}
foreach n $l {
    puts $n
}
```

変数lにリストをセットします。コマンドを実行するときは，変数置換が行われるので，変数lにセットされているリストがforeachに渡されます。

繰り返しの中では，breakとcontinueという制御コマンドが使えます。これはC言語のそれと同じ働きをします。breakは繰り返しを中断し，continueはそれ以降の処理を飛ばして次の繰り返しへ進みます。

コメント

Tclでコメントを書く場合は#を使います。コマンド名として認識される文字列が#から始まると，そこから改行までがコメントとなります。次の例を見てください。

```
%  # これはコメントです
%  # これもコメントです
% set a 10    # これはコメントではありません
エラーメッセージ
% set b 20    ;  # これはコメントになります
20
```

行の最初の文字が#の場合は，そこから改行までがコメントになります。コマンドの後ろに#を書くと，それを引数として認識するのでコメントにはなりません。ほかのプログラミング言語，たとえばPerlでは#から改行までがコメントになりますが，Tclでは違うので注意してください。セミコロン;の後ろにはコマンドを書くことができるので，コマンドの代わりに#を書くとコメントとして認識されます。

プロシージャ

C言語の関数やBASICのサブルーチンのように，Tclでは私たちユーザーがコマンドを定義することができます。これをプロシージャといい，コマンドprocを用いて定義します。

```
proc コマンド名 引数リスト 本体
```

簡単な例を示しましょう。

```
proc square {x} {expr $x * $x}
```

引数xを2乗するコマンドsquareを定義しました。引数がひとつしかない場合はリストに格納する必要はなく，引数名xだけでもかまいません。プロシージャは最後に実行したコマンドの結果を返します。この場合はexprの計算結果であるxの2乗が返り値となります。

プロシージャ内でコマンドreturnが実行されると，その引数がプロシージャの返り値となります。定義したプロシージャを呼び出す方法はいままでのコマンドとまったく同じです。

```
% square 10
100
% set a [square 20]
400
```

引数の数が足りないとエラーになります。

●局所変数

プロシージャ内で使用される変数は，プロシージャが実行されている間だけ有効です。これを「局所変数(local variable)」といいます。これとは逆に，プロシージャの外側で定義され，すべてのプロシージャからアクセスできる変数を「大域変数(global variable)」といいます。

プロシージャの引数は局所変数です。大域変数にアクセスするには，コマンドglobalを使います。次の例を見てください。

```
proc foo {a b} {
    global c
    set a [square $a]
    set b [square $b]
    set c [square $c]
    puts "a=$a, b=$b, c=$c"
}
```

プロシージャfooでは，変数a，b，cの値を書き換えていますが，a，bは局所変数でcが大域変数です。これを実行すると次のようになります。

```
% set a 10
10
% set b 20
20
% set c 30
30
% foo 100 200
a=10000, b=40000, c=900
% set a
10
% set b
20
% set c
900
```

表1 Tkに用意されている主なウィジェット

ウィジェット名	コマンド	概要
トップレベル	toplevel	新しいウィンドウを作る
フレーム	frame	ウィジェットを格納する枠組みを作る
ラベル	label	文字列やイメージを表示する
メッセージ	message	複数行の文字列を表示する
ボタン	button	ボタンを作る
ラジオボタン	radiobutton	ラジオボタンを作る
チェックボタン	checkbutton	チェックボタンを作る
リストボックス	listbox	リストボックスを作る
スクロールバー	scrollbar	スクロールバーを作る
スケール	scale	スケールを作る
エントリー	entry	1行の文字列の入力と編集
メニュー	menu	メニューを作る
メニューボタン	menubutton	メニューボタンを作る
イメージ	image	イメージを作る
ビットマップ	bitmap	ビットマップを作る
キャンバス	canvas	キャンバスを作る
テキスト	text	テキストの入力と編集

最初にa, b, cに値をセットします。この場合，プロシージャの外側なので大域変数として扱われます。次にfooを実行し，変数の値を書き換えます。実行終了後，a, bの値は変わっていませんが，cの値が変わっていますね。fooの中でa, bは局所変数なので，大域変数a, bの値には影響を及ぼしません。しかし，変数cは大域変数として扱われるので，fooの実行が終了しても値は書き換えられたままなのです。

ほかのプログラミング言語では，なにも宣言しない変数が大域変数として扱われるのが普通です。Tclでは逆になっているので注意してください。

●その他のコマンド

Tclにはこのほかにも，正規表現を含む文字列処理や，リスト操作，ファイル操作を行うコマンドが用意されています。ですが，皆さんに興味深いのはTclよりもTkのほうでしょうね。Tclの説明はこの辺で切り上げて，Tkのほうに進みましょう。Tclのコマンドは必要になったときに説明することにします。

Tkの基礎知識

Tcl/Tkの特徴は，ウィジェット(widget)と呼ばれる部品を使って簡単にGUIアプリケーションを作成できることにあります。Tcl言語だけではたいしたことはできませんが，Tkというツールキットと組み合わせることで強力な言語へと変身します。Tkに用意されている主なウィジェットには表1のようなものがあります。

なかにはあまり見かけないものもありますが，大部分はWindowsでもお馴染みのウィジェットだと思います。

ウィジェットを生成するコマンドは，Tcl言語にはもともと存在しないものです。TclからTkを制御するために拡張されたコマンドなのです。ヘルプではTk Built－In Commandsを参照してください。たくさんのウィジェットがありますが，入門編では簡単に扱うことができるウィジェットを中心に，実際にプログラムを作りながら説明していきましょう。

まず，ほとんどのウィジェットで共通するオプションを説明します(表2)。

ウィジェットの幅と高さは，テキストを表示するウィジェットでは文字数，それ以外のウィジェットはピクセル単位となります。詳しい説明は，プログラムを作りながら進めていきます。プログラミング上達への近道は，実際にプログラムを作って動作を確認することです。Tcl/Tkは手軽に実行できるインタプリタなので，動作確認も簡単に行えます。

●ボタンとラベル

まずは簡単に扱えるボタンとラベルから始めましょう。ラベルはウィンドウに文字列を表示するウィジェットです。ラベルはコマンド label で作成します。

表2 ほとんどのウィジェットで共通のオプション

オプション	説明
-foreground (-fg)	文字や線を描くのに使用する色を指定
-background (-bg)	背景色の指定
-text	ウィジェット内に表示されるテキスト
-textvariable	テキストを格納する変数
-image	ウィジェット内に表示されるイメージ
-bitmap	ウィジェット内に表示されるビットマップ
-borderwidth (-bd)	ウィジェットの枠の幅
-relief	ウィジェットの枠のスタイル
-height	ウィジェットの高さ
-width	ウィジェットの幅
-anchor	ウィジェットや表示されるデータの位置を指定

```
button    ウィジェット名 オプション
label     ウィジェット名 オプション
```

テキストを表示するウィジェットでよく使用されるオプションを示します。

```
－font        使用するフォント
－underline   下線つき表示する文字位置
－padx        水平方向の詰めもの
－pady        垂直方向の詰めもの
```

ボタンにはもうひとつ重要なオプションがあります。

```
－command   押したときに実行するコマンドを指定
```

たとえば，－commandにTcl/Tkを終了するコマンドexitを指定すると，そのボタンを押すとアプリケーションが終了することになります。

●パッケージマネージャ

ボタンを作ったら，それをウィンドウに配置しないといけません。Tcl/Tkではジオメトリマネージャ(Geometry Manager)がウィジェットの配置を担当し，3種類のマネージャが用意されています。

・Placer
placeコマンドはウィジェットを指定した座標に配置します。

・Packer
packコマンドはウィンドウにウィジェットを詰め込みます。ウィジェットの数や大きさによって，ウィンドウの大きさも変化します。

・Gridder
gridコマンドはウィジェットを格子状に配置します。このコマンドもウィジェットの数や大きさによって，ウィンドウの大きさも変化します。

いちばんよく使われるマネージャがPackerです。Placerはウィジェットの位置を座標で指定するため，並べて表示する場合には設定が少々面倒です。たいていの場合はPackerで用が足りるので，Placerを使う機会はあまりないでしょう。電卓やマインスイーパーのようにボタンを格子状に配置する場合はGridderが便利です。

それでは簡単な例題として，押したボタンの番号をラベルに表示するプログラムを作ります。最初にラベルとボタンを作ります(リスト1)。

●ウィジェット名のつけ方

ウィジェット名のつけ方は自由ですが，ピリオド(.)から始めなければいけません。単なるピリオド(.)はメインウィジェットと呼ばれ，wishが起動したときに作成されるウィンドウを表します。この中にボタンを作るので名前が.lのようにピリオドから始まります。

ウィジェット名の指定は，ファイルのパス指定とよく似ています。最初のピリオドをルートディレクトリ，ウィジェットをファイルと考えてください。

Tkではコマンドtoplevelを使って新しいウィンドウを作ることができますが，この場合もメインウィジェットから派生するウィンドウなので，名前はピリオドから始まります。たとえば新しいウィンドウを.wとすると，そこ

リスト1　ボタンとラベルの使い方

```
label .1 -textvariable buffer
pack .1
foreach n {0 1 2 3} {
    button .b$i -text "button $i" -command "push_button $i"
    pack .b$i
}
```

リスト2　ボタンを押したときの処理

```
proc push_button {n} {
  global buffer
  set buffer "押したボタンは$nです"
}
```

にボタンを作る場合，ウィジェット名は.w.b0のようになります。ピリオドがパス区切り記号と同じ役割を果たすわけです。

labelコマンドで－textvariableを指定しています。これにより，変数bufferに格納されている文字列がラベルに表示されます。これはとても便利な機能で，変数の値を書き換えるだけで表示を変更することができます。

●ウィジェットコマンドとアクション

オプション－textを使ってもいいのですが，その場合は表示を変更するときに，ウィジェットに設定されているオプションの値を変更しなければいけません。これには「ウィジェットコマンド」を使います。ウィジェットが作成されると，名前はそのウィジェットを操作するコマンドとして使うことができ，その動作を第1引数で指定します。この引数のことを「アクション」と呼びます。ほとんどのウィジェットコマンドで共通に使えるアクションがcgetとconfigureです。

ウィジェット名 cget オプション
ウィジェット名 configure 引数

cgetの場合，指定したオプションの値を取り出します。configureはオプションの値を変更するときに使います。また，引数が省略されると，設定されているオプション値をリストに格納して返します。

通常，ウィジェットコマンドとアクションを区別しなくても困ることはありません。今後は，アクションも含めてウィジェットコマンドと呼ぶことにします。

ボタンは複数作るので，ウィジェット名が重複しないように変数置換を使っています。押したときに実行するコマンドはpush_buttonです。これはprocを使って定義します。〝push_button $i″は変数置換が行われるので，設定されるボタンによって引数の値が異なることに注意してください。つまりボタン.b0が押されたときに実行されるコマンドはpush_button 0になるのです。これでどのボタンが押されたか知ることができます。

packについてはあとで詳しく説明します。残っている処理はpush_buttonです（リスト2）。

とても簡単ですね。bufferは大域変数なのでglobalコマンドを使っています。

それでは実行してみてください。ボタンが4つ表示されましたね。そして，ボタンを押すといちばん上に文字列が表示されます。つまり，ボタンを押すという動作によってプログラムが実行されたのです。

●イベントドリブンアプリケーション

GUIアプリケーションの場合，ユーザーからの入力やシステムの状態変化など，ある出来事をきっかけにプログラムが実行されます。この出来事を「イベント(event)」といい，イベントをきっかけにしてプログラムが起動されることを「イベントドリブン(eventdriven：イベント駆動)」といいます。イベントドリブン型のアプリケーションは，一般に次のようなメインルーチンを持っています。

① 初期化
② イベントを取得する
③ イベントの種類に応じて処理を振り分ける
④ ②に戻る

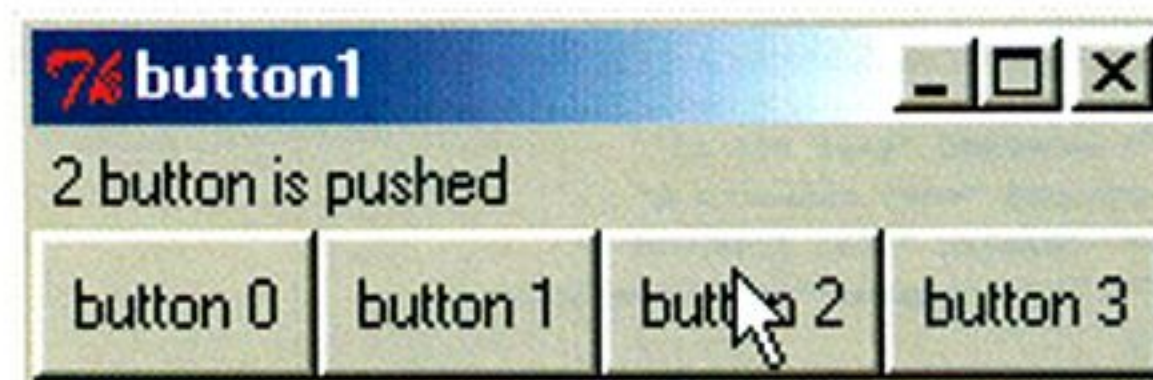



図4　ランチャプログラムもほんの数行でOK

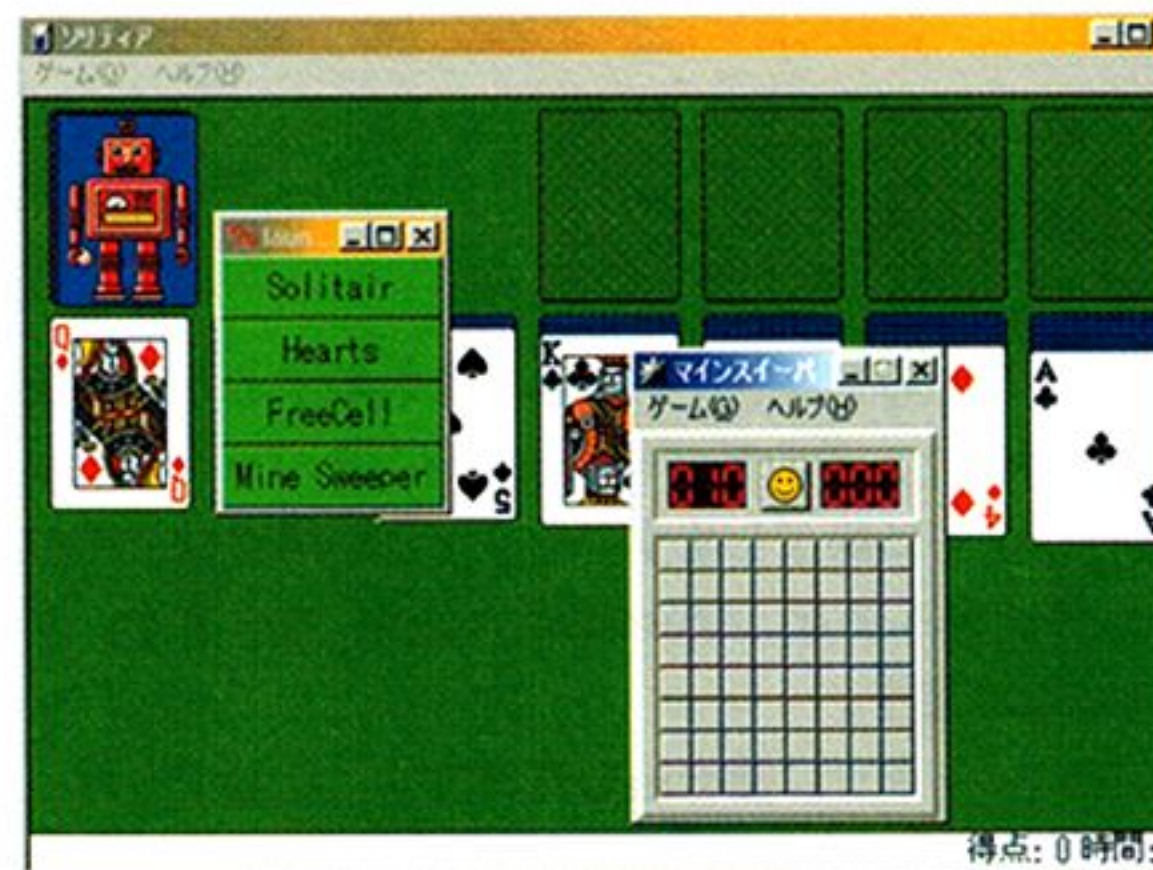



図5　ボタンに機能を割りつけてみた

②から④をイベントループと呼び，アプリケーションはユーザーからの入力などのイベントを待ちます。Tcl/Tkでプログラミングをする場合，このイベントループはインタプリタであるwishが面倒を見てくれます。

そして，③の処理に対応する機能が「バインディング(binding)」です。バインディングは，ウィンドウでイベントが発生したときに，それに応じて定義したプログラムを実行するものです。このプログラムを「イベントハンドラ」と呼びます。ボタンのオプション－commandはバインディングの一例です。このほかにもコマンドbindを使って，イベントとプログラムを関連付けることができます。

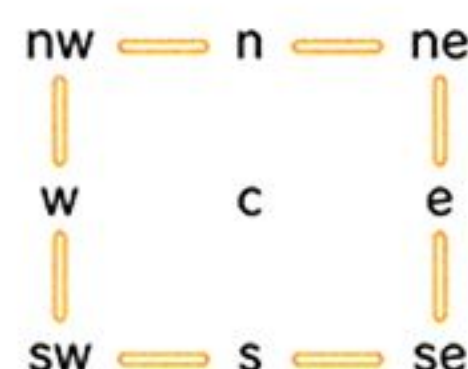
GUIアプリケーションとしての最低限の機能はwishが面倒を見てくれるので，私たちはアプリケーション固有の処理をプログラミングするだけで済みます。したがって，Tcl/Tkでのプログラミングは，ウィジェットの初期化とイベントハンドラの作成が中心となります。

●パッケージマネージャpack

次は，packについて説明しましょう。packはウィジェットを上から順に詰め込み，ウィンドウに配置するパッケージマネージャです。ボタンの幅がウィンドウより小さいですが，いっぱいに広げるには，－fillオプションを使います。方向はx，yで指定します。両方向に広げるにはbothを指定します。実際に－fillを追加して確かめてください。

詰め込む方向を変えるにはオプション－sideを使います。指定できる値はtop，bottom，left，rightの4つです。ウィジェットによって詰め込む方向を変えてもかまいません。ボタンを配置するpackに－side leftを追加して実行してみましょう。いちばん上にラベルが配置され，その下にボタンが4つ左から順番に並べられます。

このとき，ラベルはウィンドウの中央に表示されます。これを左側に寄せるには－anchorオプションを設定します。指定が省略された場合は中央となります。指定方法は次の記号を使います。

```
nw ── n ── ne
|           |
w     c     e
|           |
sw ── s ── se
```

記号はそれぞれe(East)，w(West)，s(South)，n(North)，c(Center)を表します。このほかにも，packには詰め込んだウィジェットを取り除く命令(packforget ウィジェット名)など，いろいろなオプションや命令が用意されています。

●packの実例

それでは，ちょっと実用的なツールとして，簡単なランチャを作ってみましょう。Tclにはほかのアプリケーションを起動するためのコマンドexec

リスト3　簡単なコマンドランチャ

```
button .b0 -text "ソリティア" -command "exec sol &"
button .b1 -text "ハーツ" -command "exec mshearts &"
button .b2 -text "フリーセル" -command "exec freecell &"
button .b3 -text "マインスイーパー" -command "exec winmine &"
pack .b0 .b1 .b2 .b3 -fill x
```

が用意されています。

exec アプリケーション名 引数 …… [&]

execに与える引数は，シェルからアプリケーションを起動するときと同じです。引数の最後に&をつけると，そのアプリケーションをバックグラウンドで実行し，execの実行はすぐに終了します。そうでない場合は，そのアプリケーションの実行が終了するまで，execは待つことになります。プログラムはリスト3のようになります。

ボタンを作って，packでウィンドウに詰め込むだけです。これでもれっきとしたGUIアプリケーションです。とても簡単に作れましたね。

色とスケール

ところで，ただボタンを作るだけでは面白くありません。テキストやボタンに色をつけてみましょう。Tcl/Tkの場合，色の指定は名前または数値で行います。名前はred，green，blueのように指定します。色の名前は大文字小文字の区別をしません。redとREDは同じ色を表します。使用できる色の名前はWidget Tourのデモプログラムを参照してください。数値の場合は赤，緑，青の3原色を16進数で指定します。指定方法には，次の4とおりの形式があります。

① #RGB
② #RRGGBB
③ #RRRGGGBBB
④ #RRRRGGGGBBBB

色の指定は，#から始まり，R，G，Bはそれぞれ赤，緑，青の強度を表す数値です。それぞれの色を表す桁数は同じでなければいけません。

①では，R，G，Bが16段階なので4096色の指定ができます。②は256段階なので，約1600万色の指定ができます。③④はほとんど使われることはないでしょう。実際の表示は使用しているハードウェアの環境に依存します。

●scaleウィジェット(スライダ)

それでは，R，G，Bの値で色がどのように変化するか，サンプルプログラムを作って確かめてみましょう。数値の入力はキーボードから行ってもいいのですが，ここではスケール(scale)というウィジェットを使いましょう。

スケールは整数値を表示し，スライダをドラッグするかスケールをクリックすることで，その値を更新することができます。スケールはコマンドscaleで作成します。主なオプションは次のとおりです。

－label	スケールのラベル
－from	スケールの最小値
－to	スケールの最大値
－orient	スケールの方向
－showvalue	値を表示するか
－variable	スケール値を格納する変数
－command	値が変化したときに実行するコマンド
－resolution	解像度

－labelはスケールの隣に表示する文字列を指定し，－fromと－toで値の範囲を指定します。－orientはスケールの方向を指定するもので，horizontalまたはhを指定すると水平になり，verticalまたはvで垂直になります。デフォルトでは垂直に設定されます。

－showvalueは現在の値を表示するかを設定します。－variableはスケールの値を格納する外部変数を指定します。スケールの値は，ウィジェットコマンドで求めることもできますが，外部変数を設定したほうが簡単です。－commandは，スケールの値が変更されたときに実行するコマンドを指定します。このとき，スケールの値が引数としてコマンドに渡されます。たとえば，コマンドfooを指定した場合，実行されるときはfoo 128のようになります。このほかにも，ウィジェットの大きさを設定するオプションがあります。

スケールにはconfigureやcgetのほかに，次に示すウィジェットコマンドが用意されています。

・coords 値
　指定した値に対応するスライダの座標をリストで返す
・get x y
　指定した座標に対応するスケールの値を返す。座標を省略した場合は現在値を返す
・set 値
　スケールの値を設定する
・identify x y
　指定した座標にスライダがあればsliderを返す。スライダより上(左)にあればthrough1を返し，下(右)にあればthrough2を返す

get とset以外のコマンドは使う機会はあまりないでしょう。

リスト4　ボタンとスライダの配置

```
button .b0 -text "button" -bg #000
scale .s1 -label 赤 -orient h -from 0 -to 255 ¥
   -variable red -command change_color
scale .s2 -label 青 -orient h -from 0 -to 255 ¥
   -variable blue -command change_color
scale .s3 -label 緑 -orient h -from 0 -to 255 ¥
   -variable green -command change_color
pack .b0 .s1 .s2 .s3 -fill both
```

リスト5　色の変更

```
proc change_color {n} {
   global red green blue
   set color [format "#%02x%02x%02x" $red $green $blue]
   .b0 configure -bg $color
}
```

●スケールの使用例

それでは，ボタンの背景色を変化させるプログラムを作ります。まず，ボタンとスケールを作成します(リスト4)。

スケールの値はそれぞれ，red，blue，greenという外部変数に格納します。外部変数の値は0に初期化しておきます。値が変化したときに実行するコマンドがchange_colorです。これはリスト5のように定義します。

change_colorが呼び出されるときはスケールの値が追加されるので，それを受け取る引数を定義しないとエラーになってしまいます。カラーコードの作成にはコマンドformatを使っています。このコマンドはC言語の関数sprintfと同じです。

format 書式文字列 引数 ……

formatは書式文字列の中にデータの出力形式を指定することができます。

たとえば，16進数を出力するには%xを使います。%が書式の開始を表し，xが16進数に変換することを表します。このほかに%dが10進数で，%oが8進数に変換します。%とxの間に出力する桁数(フィールド幅)などを指定することができます。簡単な使用例を示しましょう。

```
% format "<%d>" 10
<10>
% format "<%4d>" 10
<  10>
% format "<%4d>" 10000
<10000>
```

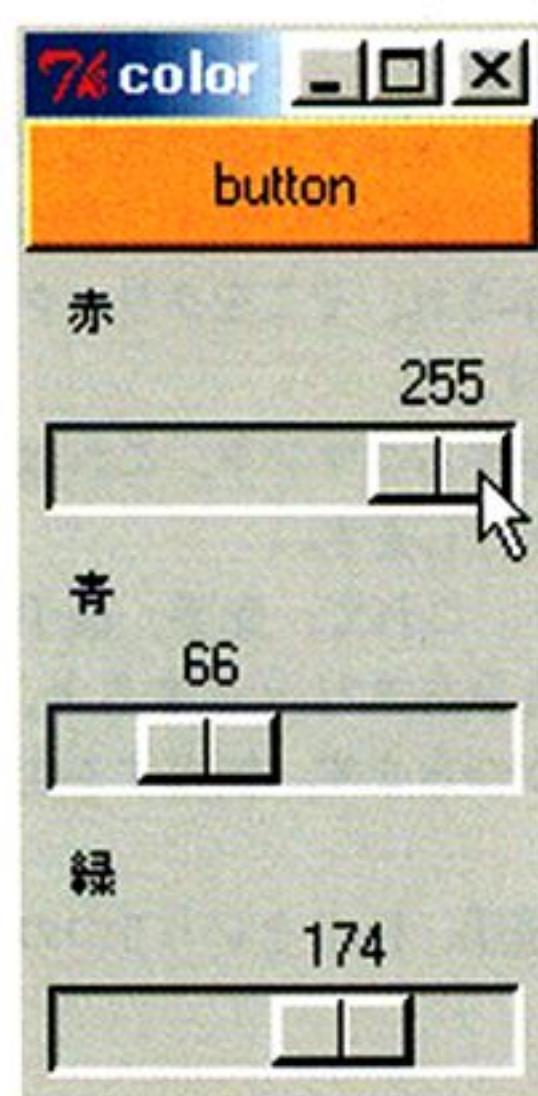



図6
スライダでRGBを指定する

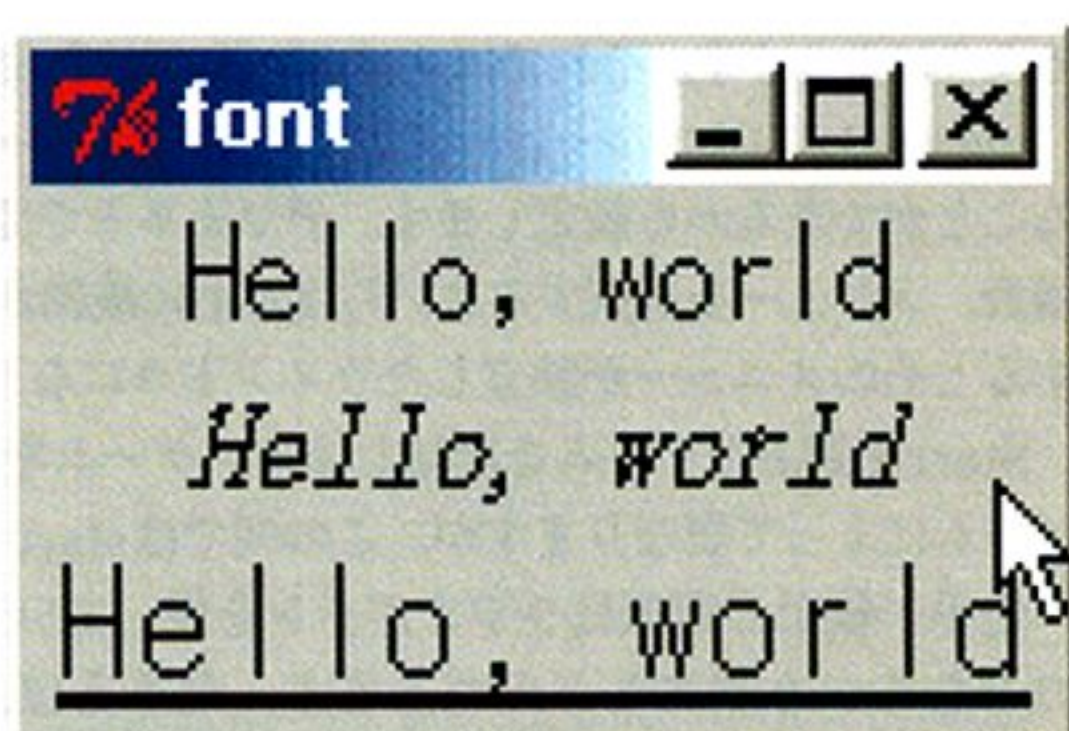



図7　フォントをいろいろ変えてみる

最初の例はフィールド幅を指定しない場合で，次の例が4に設定した場合です。10ではフィールド幅に満たないので左詰めされています。10000のように，フィールド幅に収まらない場合は指定を無視して出力します。

```
% format "<%04d>" 10
<0010>
% format "<%-4d>" 10
<10>
```

フィールド幅の前に0をつけると，左側の空いたフィールドに0を詰め込みます。－を指定すると右詰めに出力されます。

プログラムに戻ります。formatでred，green，blueの値をカラーコードに変換します。数値を2桁に揃えるため書式は%02xとしています。その後，ボタンの背景色をconfigureで変更します。これで，スライダの動きによってボタンの色を変化させることができます。

ところで，色の選択はTcl/Tkで便利なコマンドtk_chooseColorが用意されています。このコマンドを実行すると，色を選択するためのウィンドウが開かれます。Widget Tourにデモプログラムがありますので，そちらを参照してください。

フォントの指定

今度は色だけではなくフォントも変更してみましょう。フォントの指定には，いくつかの方法があるのですが，Windows上ならば次の形式で行えばいいでしょう。

```
family size style1 style2
```

familyはフォント名を表します。いまあなたが使っているパソコンで使用できるフォント名は，コマンドfont familiesで求めることができます。fontはフォントを操作するためのコマンドです。Windowsであれば{MS 明朝}や{MS ゴシック}といった名前を見つけることができるでしょう。sizeはフォントの大きさを表し数値で指定します。style1とstyle2はフォントのスタイルで，次の中から選びます。

```
style1 normal, bold, roman, italic
style2 underline, overstrike
```

style1とstyle2は省略することができます。それでは，フォントを変更してみましょう。リスト6のプログラムを実行してみてください。

テキストを表示するウィジェットはオプション－fontで使用するフォントを指定することができます。

このように，個々のウィジェットのフォントはこれで変更できますが，すべてのラベルウィジェットで使用する共通のフォントを設定したい場合もあるでしょう。Tkは各オプションのデフォルト値を持っています。このため，

リスト6　フォントの変更

```
label .10 -text "Hello, world" -font {{MS ゴシック} 12}
label .11 -text "Hello, world" -font {{MS 明朝} 12 italic}
label .12 -text "Hello, world" ¥
                -font {{MS ゴシック} 16 underline}
pack .10 .11 .12
```

ユーザーは必要なオプションを指定するだけで，簡単にプログラミングすることができました。このデフォルト値はコマンドoptionを使って変更することができます。たとえば，アプリケーションで使用するフォントを変更する場合は，次のように行います。

```
option add *font {{MS ゴシック} 12}
```

これで，テキストを表示するウィジェットは，指定したフォントを使って表示されます。第2引数は値を設定するウィジェットを表すパターンです。パターンは，アプリケーション名，ウィジェット名，オプション名をドットで区切って表しますが，ワイルドカード＊やウィジェットを表すクラス名を指定することもできます。クラス名は，そのウィジェットを表す型名と考えてください。

＊fontの場合はwish.fontと同じ意味で，アプリケーションwishで使用するフォントを指定することになります。ラベルに対してフォントを設定したい場合は，＊Label.fontとなります。たいていのクラス名は，ウィジェットを生成するコマンドの，先頭の文字を大文字にしたものです。詳しくはヘルプを参照してください。

たとえば，ランチャプログラムの先頭に次の2行を加えてください。

```
option add *Button.font {{MS ゴシック} 12}
option add *Button.background green
```

これで，表示されるボタンのフォントと背景色は，設定された値となります。

メニューバー

次はGUIには欠かせないメニューの作り方を説明します。Tcl/Tkではメニューのためのコマンドがたくさん用意されていて，いろいろなメニューを構成することができますが，入門編はWindowsでも標準になっている「メニューバー」という方法を説明します。

● menu

メニューを作るにはコマンドmenuを使います。

```
menu ウィジェット名 オプション
```

メニューバーの場合はオプション－typeでmenubarを指定します。作成したメニューバーはウィンドウを作成するコマンドtoplevelのオプション－menuで設定します。メインウィジェット「.」の場合はconfigureを使って変更します。

```
menu .m -type menubar
. configure -menu .m
```

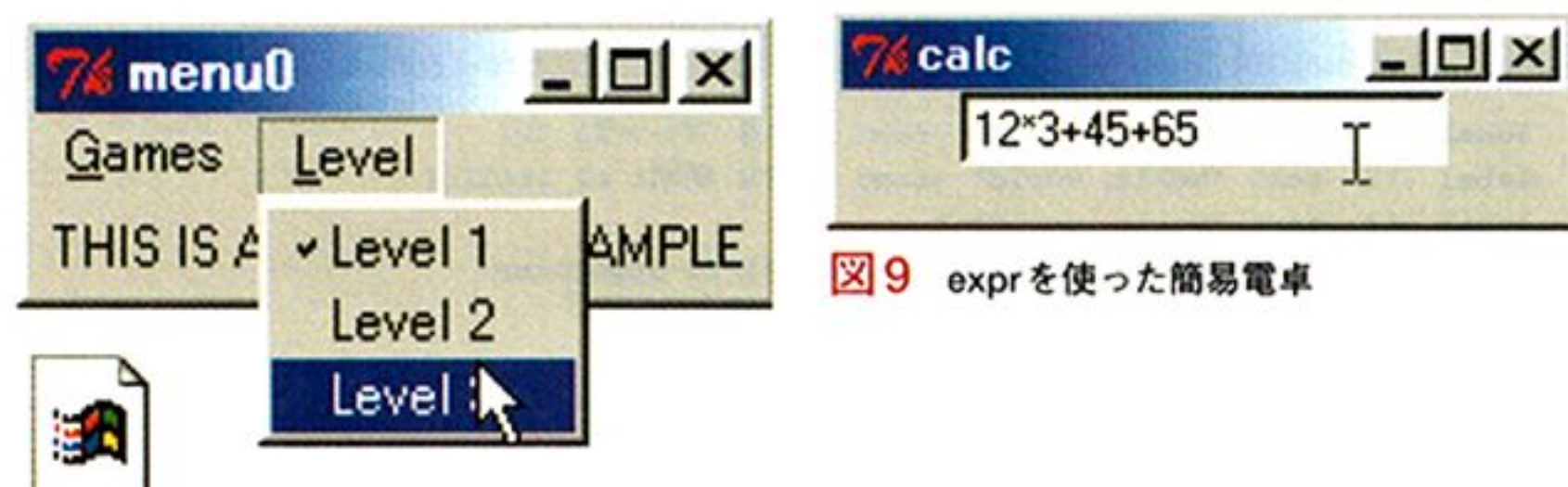
これでウィンドウにメニューバーが設定されました。あとはこのメニューバーに具体的なメニューを追加します。これにはウィジェットコマンドaddを使います。

```
.m add 項目 オプション
```

第2引数の項目には，具体的なメニューを指定します。

・cascade

図8　メニューも数行で作成できる

図9　exprを使った簡易電卓

リスト7　Gamesメニューを作る

```
menu .m.m1 -tearoff no
.m.m1 add command -label "Start" -underline 0 -command "start"
.m.m1 add separator
.m.m1 add radiobutton -label "先手" -variable action -value 0
.m.m1 add radiobutton -label "後手" -variable action -value 1
.m.m1 add separator
.m.m1 add command -label "Exit" -underline 0 -command "exit"
```

リスト8　Levelメニューの作成

```
menu .m.m2 -tearoff no
.m.m2 add radiobutton -label "Level 1" -variable level -value 1
.m.m2 add radiobutton -label "Level 2" -variable level -value 2
.m.m2 add radiobutton -label "Level 3" -variable level -value 3
```

複数のメニューを表示する

- checkbutton
  チェックボタンを表示
- commandオプション
  −commandで指定したコマンドを実行
- radiobutton
  ラジオボタンを表示
- separator
  区切りを表示する

cascadeを指定すると，そのメニューを選択したときに複数のメニューを表示します。checkbuttonはyes/noのような二者択一の情報を設定するために使います。commandはメニューが選択されたときに，オプション−commandで指定したコマンドを実行します。radiobuttonは複数の値からひとつを選ぶ場合に使います。separatorは区切りを表示するだけです。

checkbuttonとradiobuttonはメニューバーに直接定義するのではなく，cascadeと組み合わせて使うことが一般的です。checkbuttonとradio buttonを使う場合，選択する値をオプション−valueで指定し，その値を格納する変数をオプション−variableで指定します。また，−commandオプションを設定することもできます。この場合，変数に値がセットされるとともに，指定したコマンドが実行されます。

たとえば，将棋やリバーシのようなゲームのメニューを考えてみましょう。最低限必要となるメニューは，ゲームの開始，先手と後手の選択，コンピュータの強さの設定，などでしょうか。最初の2つはメニューGamesで設定し，強さはメニューLevelで選択することにします。この場合，まず，GamesとLevelをメニューバーに追加します。

```
.m add cascade −label "Games" −underline 0 −menu
.m.m1
.m add cascade −label "Level" −underline 0 −menu
.m.m2
```

オプション−underlineは，ラベルの文字に下線を付け加えます。Windowsの場合，Altキーでメニューを選択できますが，この状態で下線のついた文字をキーボードから入力することで，そのメニューを選ぶことができます。

cascadeを使う場合，表示するメニューをオプション−menuで設定します。このメニューはメニューバー.mの中に設定するので，ウィジェット名は.m.m1のようになります。それではメニューGamesを設定します(リスト7)。

menuのオプション−tearoffは，そのメニューをウィンドウから引きちぎることができるかを設定します。デフォルトではyesになっています。その場合，メニューを選択するといちばん上に破線が表示され，そこをクリックするとそのメニューが独立したウィンドウになります。

Startを選ぶとゲームを開始します。ゲームを開始するコマンド，これはゲームによって異なりますが，この例ではstartを実行します。

先手・後手の選択はラジオボタンを使っています。これで，先手，後手のどちらかを選ぶことができます。たとえば，後手をクリックすると，actionの値は1にセットされ，ラベルの左側にレ点がつきます。使用する変数は，あらかじめ初期化しておきましょう。

これでGamesをクリックすると，Start，先手・後手，Exitという3つのメニューが現れます。

次はメニューLevelの設定です。

ラジオボタンを使えば3つの中からひとつを選ぶことができます。ゲームの中身は空ですが，このように簡単にメニューを設定することができます。

チェックボタンとラジオボタンは，メニューだけではなくウィジェットとして生成することができます。コマンドはcheckbuttonとradiobuttonで，使うオプションはメニューの場合と同じです。Widget Tourにデモプログラムがありますので，参考にしてください。

キー入力とバインド

いままでの例題は，マウスで操作するものばかりでした。今度はキーボードからの入力を受け付けるウィジェットを説明します。

●エントリー

エントリー(entry)は1行の文字列を入力，または編集することができます。例題として，数式を入力して計算するcalc.tclを作ります。これはとても簡単に作ることができます。まずエントリーから説明しましょう。

```
entry ウィジェット名 オプション
```

entryでよく使うオプションは−textvariableです。entryで入力されたデータは指定した大域変数に格納されます。また，大域変数の値が変更されると，entryの内容も変更されます。面白いオプションが−showです。これはパスワードのように画面に見えてはいけない文字列を打ち込むときに使います。たとえば，−show"*"とすれば，入力された文字は*として表示されます。

ウィジェットコマンドにはcget，configure のほかに，文字列の取得，挿入，削除，カーソルの移動，カット&ペースト，スクロールなど，たくさん用意されていますが，文字列の入力だけならば，それらを使う機会はあまりないでしょう。また，エントリーのキー操作はEmacsに準じているので，Emacs/Muleを使っているユーザーには扱いやすいと思います。

●式入力電卓の制作

それではプログラムを作りましょう。データの入力が完了したらボタンを押してもらってもいいのですが，データはキーボードから入力するのですから，マウスよりもキーボードで操作したほうがいいでしょう。リターンキーの入力でデータを計算するようにします。キー入力もイベントのひとつですからバインディングを設定することができます。プログラムは次のようになります。

```
entry .e0 −textvariable buffer
pack .e0

bind .e0 <Return> {
    set buffer [expr $buffer]
}
```

計算はexprを使えばいいので簡単ですね。といっても，Tclの数学関数が使えるので，sin，cos，tanなど関数電卓としても使うことができます。

●イベント処理

このプログラムのポイントはbindコマンドです。

```
bind ウィジェット名 イベント [+]コマンド
```

すでにバインドされているコマンドがある場合，新しいコマンドに差し替えられます。コマンドの前に+をつけると，既存のコマンドに新しいコマンドが追加されます。コマンドを省略すると，そのイベントにバインドされているコマンドが返されます。ウィジェット名だけを指定すると，そのウィジェットにバインドされているイベントが返されます。

イベントの指定は次のような構文を持っています。

```
< modifier − modifier − type − detail >
```

typeはGUI環境上で発生するイベントタイプを表します。ユーザーが操作するときに発生する主なイベントタイプには次のようなものがあります。

Key，KeyPress	キーが押された
KeyRelease	キーが離された
Button，ButtonPress	マウスのボタンが押された
ButtonRelease	マウスのボタンが離された
Motion	マウスの移動
Enter	マウスカーソルがウィンドウの中に入った
Leave	マウスカーソルがウィンドウから出た

このほかにも，ウィンドウが破棄されたときに発生するイベントなど，さまざまなイベントタイプがあります。

マウスとキーのイベントには，ボタンやキーの種類をdetailで指定します。マウスでは左ボタンが1となります。キーの種類は名前(Keysym)で指定します。英数字はその文字がそのまま名前となります。このほかに，改行キーに対するReturn，バックスペースキーに対するBackSpaceなどがあります。

detailを指定する場合はtypeを省略することができます。ただし，<1>という指定は<KeyPress-1>ではなく<Button-1>となるので注意してください。また，通常の英数字の場合，<>も省略することができます。つまり，<KeyPress-a>はaと書くことができます。それから，<KeyPress>のようにdetailを省略すると，種類によらずキーが押されたときにバインドされたコマンドが実行されます。

●モディファイア

イベントタイプの前にはモディファイア(modifier)をつけることができます。たとえば，<Control-d>はコントロールキーとdを同時に押したときのイベントを表します。主なモディファイアを次に示します。

Control	Ctrl キーを押しながらの入力
Shift	Shift キーを押しながらの入力
Alt	Alt キーを押しながらの入力
Button1，B1	マウスの左ボタンを押しながらの入力
Button3，B3	マウスの右ボタンを押しながらの入力
Double	ダブルクリック
Triple	トリプルクリック

Tcl/Tkの出身地であるX Windowでは3ボタンマウスを使うので，Button2は右ボタンではなく中ボタンとなります。たとえば，左ボタンのダブルクリックに対応するイベントは<Double −1>となります。また，イベントタイプは複数個指定することができます。たとえば，<Escape>aはEscキーが押されたあとでキーaを押したイベントに対応します。

バインドされたコマンド内では，イベント情報を取得するための方法が用意されています。% から始まる文字列はイベント情報に置換されます。

%b	マウスボタンの番号
%x，%y	マウスカーソルの座標
%W	ウィンドウのパス名
%A	キーに対応する文字
%K	キーに対応する名前(Keysym)
%%	% 自身を表す

置換の指定は30種類以上ありますので，詳細はヘルプを参照してください。たとえば，次のプログラムをwishのコンソール上から実行すると，キーに対応する名前を表示することができます。

```
bind . <KeyPress> {puts "Keysym is %K"}
```

実際に試してみると，F1やF2キーにはF1，F2という名前が割り当てられていることがわかります。

リストボックスとスクロールバー

次は，リストボックスとスクロールバーというウィジェットを説明します。リストボックスは複数の文字列を表示し，ユーザーはそのなかからひとつ以上の文字列を選ぶことができます。スクロールバーは，ほかのウィジェットの表示範囲を制御します。例題として，calc.tclで入力した計算式をリストボックスに格納しておいて，必要なときに取り出せるように改造してみましょう。最初にリストボックスから説明します。

```
listbox ウィジェット名 オプション
```

listboxで指定する主なオプションは，表示範囲のコントロールと選択方法です。

−xscrollcommand	x 方向のスクロールコマンドを指定
−yscrollcommand	y 方向のスクロールコマンドを指定
−selectmode	セレクションモード

スクロールコマンドは，リストボックスの表示範囲が変更されたときに呼び出されるコマンドです。たとえば，スクロールバーと連動させる場合，このコマンドはスクロールバーの位置を動かすウィジェットコマンドsetが指定されます。これはコマンドscrollbarと一緒に説明します。セレクションには次のモードが用意されています。

・single
ひとつの行をマウスの左クリックで選択する。

・browse
single と同じだが，ドラッグによって選択される行が変化し，ボタンを離したところの行が選択される。

・multiple
左クリックで複数行を選択する(ドラッグは不可)。

・extend
ドラッグで複数行を選択するが，左クリックではいままで選択した行はキャンセルされ，クリックした行のみ選択される。Ctrlキーを押しながら左クリックするとトグル動作(結果が反転)となり，シフトキーを押しながら左クリックすると直前に左クリックした行から現在の行までが選択される。

−selectmodeのデフォルト値はbrowseです。データの挿入，削除，取得はウィジェットコマンドで行います。

insert 位置 文字列 ……	指定した位置の直前に文字列を挿入
delete first last	指定した範囲の行を削除する

get first last	指定した範囲の行をひとつの文字列として返す
index 位置	指定した位置の行番号を返す
curselection	選択された行番号をリストにして返す
see 位置	指定した位置が見えるようにスクロールする

このほかにもいろいろなコマンドがありますが，特にスクロールバーに関係するxview/yviewコマンドが重要です。これは，scrollコマンドのところで説明します。

位置の指定には次の方法があります。

- n（数値）
  n行目
- active
  左ボタンを離したときの行
- anchor
  左ボタンを押したときの行
- end
  最後の行，ただし，insertで指定すると最終行の次にデータが追加される
- @x，y
  指定した座標にもっとも近い行

セレクションモードがextendのときにドラッグで選択した場合，最初の行がanchorで最後の行がactiveとなります。したがって，deleteにanchorとactiveを指定すると，選択した行をリストボックスから削除することになります。

●スクロールバー

次はスクロールバーを説明します。スクロールバーは，その両端に矢印がつき，中央付近には四角いスライダが表示されます。矢印を左クリックするか，スライダをドラッグすることで表示位置を変更します。また，矢印とスライダの隙間をクリックすると1画面分スクロールします。スクロールバーを作るコマンドはscrollbarです。

scrollbar ウィジェット名 オプション

scrollbarで主に使用されるオプションには次のものがあります。

−orient	スクロールバーの方向
−throughcolor	矢印とスクロールの隙間の色
−command	スクロールバーが動いたときに実行するコマンド

−orientはスケールと同じくスクロールバーの方向を指定するもので，horizontalまたはhを指定すると水平になり，verticalまたはvで垂直になります。−commandはスクロールバーを動かしたときに実行するコマンドを指定します。リストボックスとスクロールバーを連動させる場合，このコマンドはリストボックスの表示位置を制御するコマンドxviewやyviewが指定されます。

●setコマンド

スクロールバーで重要なウィジェットコマンドはsetです。

set first last スライダの位置を指定する

setの引数firstとlastは0から1の間の実数で，表示されている範囲を表しています。たとえばリストボックスと連動している場合，全体の行数が100行で20行目から30行分表示されているとすると，set 0.2 0.5となります。つまり，データ全体の20%の位置から50%の位置まで表示されていることを表します。スクロールバーではこのデータからスライダの位置と大きさを調整します。

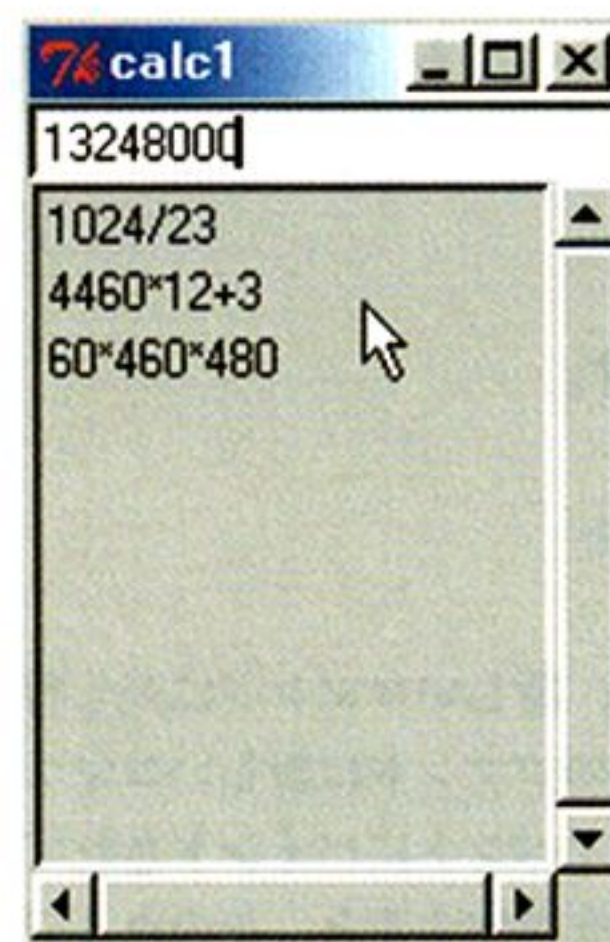



図10
式の履歴が残るように改造した電卓

リスト9　ウィジェットの追加

```
entry .e0 -textvariable buffer
listbox .lb -yscrollcommand ".s1 set" \
   -xscrollcommand ".s2 set"
scrollbar .s1 -orient vertical -command ".lb yview"
scrollbar .s2 -orient horizontal -command ".lb xview"
grid .e0 -row 0 -column 0 -columnspan 2 -sticky ew
grid .lb -row 1 -column 0 -sticky nsew
grid .s1 -row 1 -column 1 -sticky ns
grid .s2 -row 2 -column 0 -sticky ew
```

リスト10　データの選択

```
bind .e0 <Return> {
   .lb insert end $buffer
   .lb see end
   set buffer [expr $buffer]
}

bind .lb <Double-1> {
   set buffer [.lb get active]
}
```

setはリストボックスのオプション−xscrollcommandや−yscrollcommandで指定します。スクロールバーのウィジェット名を.sとすると，指定方法は次のようになります。

−xscrollcommand ".s set"

リストボックスで表示範囲が変更されるとスクロールコマンドが実行されますが，このとき，リストボックスの表示範囲が引数として付け加えられ，コマンドが実行されます。

setはリストボックスの変更をスクロールバーに反映させるために使いましたが，スクロールバーの変更をリストボックスに反映させるためのオプションが−commandです。ここにリストボックスのウィジェットコマンドxviewやyviewを指定します。指定方法は簡単で，リストボックスのウィジェット名を.lbとすると次のようになります。

listbox .lb −yscrollcommand ".s set"
scrollbar −orient v −command ".lb yview"

スクロールバーの操作によって，次に示す文字列が付け加えられてコマンドが実行されます。

- moveto 数値
  指定した数値（0−1.0）の位置までスクロール
- scroll [＋¦−]1 unit
  上または下に1単位スクロールする
- scroll [＋¦−]1 pages
  上または下に1ページスクロールする

まあ，付け加えられるデータを無理に覚える必要はありません。スクロールバーを使うときは，連動するウィジェットのスクロールオプションにsetを指定して，スクロールバーの−commandに表示を制御するウィジェットコマンドを指定する，と理解しておけば十分でしょう。

図11 GIFイメージを表示してみた

●電卓の改造

それでは，calc.tclを改造しましょう。まず必要なウィジェットを生成します。

リストボックスとスクロールバーはgridで配置します。gridは格子状にウィジェットを配置するジオメトリマネージャです。ウィンドウをM行N列のセルに分割し，そこにウィジェットを配置するのです。x方向の位置はオプション－columnで指定し，y方向の位置は－rowで指定します。gridにはpackとは違うオプション－columnspanと－rowspanがあります。これは，複数のセルにまたがってウィジェットを配置するために使います。－columnspanはx方向にまたがるセルの数，－rowspanはy方向にまたがるセルの数を指定します。それから，packではオプション－fillでウィジェットを引き伸ばすことができましたが，gridではオプション－stickyを使います。

```
{ }     上下左右とも中央寄せ
n       上寄せ
s       下寄せ
e       右寄せ
w       左寄せ
ns      上下方向に引き伸ばす
ew      左右方向に引き伸ばす
```

－stickyはpackのオプション－anchorと同じ機能もあわせ持っています。

エントリーはいちばん上に配置しますが，－columnspanでx方向にセルをつなげて，－sticky ewで左右に広げています。

●バインディング設定

次はバインディングを設定します。リストボックスからデータを選ぶ処理ですが，ダブルクリックしてもらうことにします。

エントリーではリターンキーが入力されたら，データをリストボックスに代入し，exprの計算結果をbufferにセットします。これが逆になると，答えをリストボックスに代入することになります。それから，ウィジェットコマンドseeを使って，セットした計算式が見えるようにスクロールしています。

リストボックスでは，ダブルクリックされた位置からgetでデータを取り出してbufferにセットします。ダブルクリックですから位置はactiveとanchorどちらでもかまいません。

イメージ

今度は画像の取り扱いについて説明しましょう。以前のTkでは，白黒のビットマップしか扱えませんでしたが，ver.4.0以降のTkでは標準でカラーイメージをサポートし，GIFやPPM/PGM形式の画像ファイルを扱うことができるようになりました。PPMはカラー，PGMはグレイスケールの画像を扱う，UNIXで標準的に用いられるベタフォーマットです。

Tkでは，コマンドimageでイメージを作り出し，そのイメージをラベルやボタンなどのウィジェットに使うことができます。たとえば，画像ファイルからイメージを作るには，次のように行います。

```
image create photo 名前 －file ファイル名
```

createはイメージの生成，photoはイメージの種別でカラーを意味します。名前は生成したイメージを表します。省略した場合は適当な名前がつけられ，その名前がimageの返り値となります。imageはcreate以外にも次の操作を行うことができます。

```
image delete 名前       イメージを削除する
image types             指定可能な種別のリストを返す
image names             全イメージの名前をリストにして返す
image type 名前         イメージの種別を返す
image height 名前       イメージの高さを返す
image weight 名前       イメージの幅を返す
```

ラベルやボタンにイメージを表示するには－imageオプションを使います。

```
label .l0 －image 名前
```

これでラベルにイメージが表示されます。それでは簡単な例題として，GIF/PPMの画像を表示するプログラムを作ります。このようなアプリケーションの場合，ファイルの指定方法がGUIらしくないとボロクソにいわれるものですが，幸いなことにTkにはファイルを選択するためのコマンドが用意されています。

```
tk_getOpenFile    入力ファイルを選択
tk_getSaveFile    出力ファイルを選択
```

これらのコマンドを実行すると，ファイル選択のウィンドウ(ダイアログ)が開かれ，ウィンドウ上の操作でディレクトリをたどり，ファイルを選ぶことができます。使用できるオプションは次のとおりです。

－initialdir ディレクトリ
　最初に選択されているディレクトリ
－initialfile ファイル
　最初に選択されているファイル(出力ファイルのみ有効)
－defaultextension 拡張子
　最初に選択されている拡張子
－filetypes パターンリスト
　使用可能なファイル種別と拡張子のリストを指定
－parent ウィンドウ
　ダイアログボックスの親ウィンドウを指定
－title 文字列
　ダイアログボックスのタイトル

このなかで重要なオプションが－filetypesです。アプリケーションで扱うことができるファイル種別を拡張子で指定し，そのファイルだけを表示します。指定はリストで行います。

```
－filetypes { ファイル種別 …… }
ファイル種別 := {{名前} {拡張子 …… }
```

たとえば，GIF/PPMファイルを指定する場合は，次のようになります。

```
－filetypes {
    {{画像Files} {.gif .ppm}}
}
```

この場合はGIFとPPMファイルが一緒に表示されます。次のように指定すると，表示するファイルをダイアログの操作で切り替えることができます。

リスト11　画像ローダ用メニュー

```
menu .m -type menubar
. configure -menu .m
.m add cascade -label "File" -under 0 -menu .m.m1
menu .m.m1 -tearoff no
.m.m1 add command -label "Open" -under 0 -command "load_file"
.m.m1 add separator
.m.m1 add command -label "Exit" -under 0 -command "exit"
```

リスト12　グローバル変数の定義

```
set path_name ""
image create photo image_data -width 64 -height 64
label .10 -image image_data
pack .10
```

リスト13　ローダ本体

```
proc load_file {} {
   global image_data path_name
   set filename [tk_getOpenFile -initialdir $path_name \
          -filetypes {{{画像Files} {.gif .ppm}}}]
   if {$filename != ""} {
       set path_name [file dirname $filename]
       image delete image_data
       image create photo image_data -file $filename
       .10 configure -image image_data
   }
}
```

```
-filetypes {
   {{GIF Files} {.gif}}
   {{PPM Files} {.ppm}}
   {{ALL Files} {*}}
}
```

すべてのファイルを表示する場合は*を使います。また，空文字列""を指定すると，拡張子のないファイルを表示します。ファイルを選択すると，ファイル名をフルパス形式で返します。選択しない(キャンセル)場合は，空文字列が返されます。

●GIF/PPMローダの制作

それでは，GIF/PPM画像ローダを作りましょう。まず，メニューから設定します。

メニューFileの下に，ファイルを選択するOpenとアプリケーションを終了するExitの2つのメニューを設定します。次に，イメージとグローバル変数を定義します。

path_nameは選択されたファイルのパスを格納しておきます。tk_getOpenFileにこのパスを指定することで，次にファイルを選ぶときは同じディレクトリから始めることができます。アプリケーションの開始時にはファイルは指定されていないので，空のイメージを作って表示しておきます。あとは，画像ファイルをロードする本体を作ります。

tk_getOpenFileでファイル名を取得したら，パスを取り出してpath_nameにセットします。fileはファイル情報を取得するTclのコマンドです。filedirnameはファイル名からパス部分を返します。tailを指定するとパス部分を取り除いたファイル本体の名前を返します。このほかにも，ファイルの種別や時刻などさまざまな情報を得ることができますが，詳細はヘルプを参照してください。

ファイル名をゲットしたら，それが空文字列でないことを確認します。次に，表示しているイメージをimage deleteで削除してから，新しいイメージをimage createで生成します。最後に，ラベルのconfigureで表示するイメージを変更します。とても簡単ですね。

ところで，イメージを生成するには，ファイルからロードするだけではありません。イメージを直接操作することができます。いろいろなコマンドが用意されていますが，入門編では説明を割愛させていただきます。興味のある方は参考文献やヘルプをあたってみてください。

キャンバスウィジェット

最後に図形を表示する「キャンバス(canvas)」ウィジェットを説明します。キャンバスは，矩形，直線，楕円などの図形のほかに，イメージ，文字列，任意のウィジェットを表示することができます。まず，キャンバスウィジェットを生成してみましょう。wishのコンソールから次のコマンドを打ち込んでください。

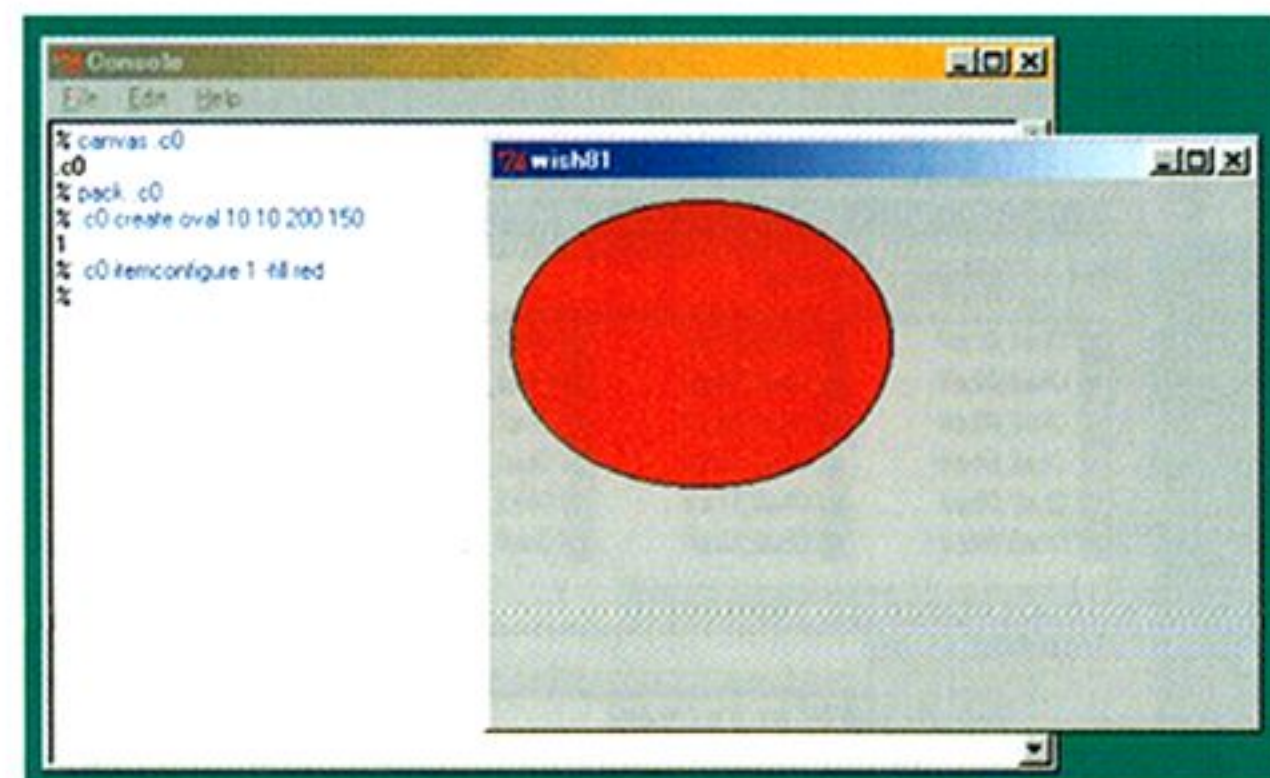



図12　キャンバスに図形を描画する

```
% canvas .c0
.c0
% pack .c0
```

空のウィンドウが表示されました。これで図形を表示するキャンバスをウィンドウに配置したことになります。また，キャンバスとスクロールバーを組み合わせて，表示範囲を変更することもできます。

●createコマンド

キャンバスを配置しただけでは，なにも図形は描かれていません。図形を生成するには，ウィジェットコマンドcreateを使います。

```
create 種別 座標 オプション
```

指定できる種別には，次のものがあります。

line	直線(折れ線)
oval	楕円
arc	円弧(楕円の円周の一部)
rectangle	矩形
polygon	多角形
image	イメージ
bitmap	ビットマップ
text	文字列
window	任意のウィジェット

●楕円を描く

それでは実際に図形を表示してみましょう。

```
% .c0 create oval 10 10 100 100
1
```

ウィンドウに楕円が描画されました。楕円の場合，指定した矩形に内接するように描画されます。返された数値は図形を表す番号(ID)です。これを使って図形を操作することができます。ウィジェットコマンドのcgetやconfigureに対応するのが，itemcgetとitemconfigure です。たとえば，楕円の中を赤色に塗りつぶしてみましょう。

```
% .c0 itemconfigure 1 -fill red
%
```

楕円の中が赤くなりましたね。よく使われるオプションには次のものがあります。

```
－fill 色                内部を塗りつぶす色
－stipple ビットマップ  内部を塗りつぶすときの模様になるビットマップ
－outline 色             枠の色
－width 幅               枠の幅（デフォルトは1.0）
```

－fillのデフォルトはnoneです。これは透明を表してます。－fillで色を指定した後でも，itemconfigureでnoneに再設定すれば透明に戻すことができます。

●矩形を描く

矩形も楕円と同じ指定方法です。－stippleには，wishに標準で組み込まれているビットマップを指定するのが一般的です。よく使うビットマップが灰色の模様を表すgray12，gray25，gray50，gray75です。Widget Tourにデモプログラムが用意されているので見てください。それでは実際に描画してみましょう。

```
% .c0 create rectangle 10 100 100 200 －fill green
                                    －stipple gray25
2
```

●直線を描く

次は直線です。2点間だけではなく複数の点を指定すると，その間を直線で結びます。直線を描画する前に，楕円と矩形を消しましょう。

```
% .c0 delete 1 2
```

これで空のウィンドウに戻りました。deleteは図形を消去するcarvasのウィジェットコマンドです。では，直線を描画してみましょう。

```
% .c0 create line 10 10 200 10 10 200 200 200
3
```

画面にZ字形の線が描かれましたね。線の色を指定するオプションは，直線の場合は－outlineではなくて－fillで指定します。では緑色に変更してみましょう。

```
% .c0 itemconfigure 3 －fill green －width 2.0
%
```

－widthで線を太くしています。オプション－smoothをtrueに指定すると，滑らかな曲線を描画することができます。

```
% .c0 create line 10 10 200 10 10 200 200 200
                              －smooth true －fill red
4
```

このほかにも，矢印の設定や折り返し時の形など，いろいろなオプションが用意されています。

●多角形を描く

次は多角形です。五角形を作ってみましょう。各頂点の座標を指定しますが，最初の点と最後の点が結ばれて閉じた図形となります。

```
% .c0 create polygon 10 60 60 10 110 60 85 150 35 150
5
```

polygonでは，デフォルトで－fillオプションが黒，－outlineは描画されません。それから，lineと同様に－smoothをtrueに指定すると，多角形の角を丸めます。実際に試してみてください。

●円弧を描く

次は円弧です。楕円の円周の一部分を表示します。座標の指定はovalと同じですが，オプションで表示する範囲を指定します。

```
－start 角度   開始位置を角度で指定
－extent 角度  終了位置を開始位置からの角度で指定
－style 種別   arc：円周のみ描画，
               chord：円周と始点終点を結ぶ線分
               pieslice：円周と中心から始点，終点を結ぶ線分
```

角度は度数でプラスが反時計回り，マイナスが時計回りとなります。また，ovalと同じオプションが使えます。ただし，－styleがarcの場合，－fillで色を指定しても表示されません。chordかpiesliceに変更すると表示されます。

●イメージの表示

キャンバスはイメージとビットマップも表示することができます。

```
image x y オプション
bitmap x y オプション
```

x，yは表示する座標を表します。イメージのどの位置に対応させるかは，オプション－anchorで指定します。これはpackと同じ指定方法です。データとの対応は－imageと－bitmapで指定します。それでは，例題としてTcl/Tkの配布キットにあるロゴ（logomed.gif）をキャンバスに表示してみましょう。

```
% image create photo i0 －file logomed.gif
i0
% .c0 create image 100 100 －image i0
6
```

これでキャンバスにイメージが描画されます。

●文字列の表示

次は文字列です。当然ですがキャンバスに文字を描くことができます。

```
text x y オプション
```

x，yは座標で，オプションには次のものが使えます。

```
－anchor 位置    座標とテキストの位置関係
－font フォント  文字のフォント
－fill 色        文字の色
－justify mode   center:中揃え，left:左揃え，right:右揃え
－text 文字列    表示する文字列
－width 長さ     1行の長さ
```

それでは実際に試してみましょう。

```
% .c0 create text 10 10 －text "hello, world!" －anchor nw
7
```

これで（10，10）の位置からhello，world!が表示されます。

●ウィジェットの挿入

キャンバス中にほかのウィジェットを表示させる場合はwindowを使います。

```
－anchor 位置        座標とテキストの位置関係
```

－window ウィジェット　表示するウィジェット
－width 幅　ウィジェットの幅
－height 高さ　ウィジェットの高さ

たとえば，ラベルを表示させてみましょう。

```
% label .l0 －text "hello, world!" －bg green
.l0
% .c0 create window 30 30 －window .l0 －anchor nw
8
```

今度は背景色が緑のhello，world!が表示されました。このほかにも，Widget Tourにはいろいろなデモプログラムが用意されているので，参考にしてください。

●図形操作用コマンド

キャンバスで使用できる図形をひととおり説明したところで，図形を操作するときによく使うコマンドを説明しましょう。

type ID	図形の種別を返す
bbox ID……	指定した図形を囲む領域（矩形）をリストにして返す
coords ID [座標……]	図形の座標の設定や問い合わせ
move ID dx dt	図形の移動
lower ID [ID]	重なり順を低くする
raise ID [ID]	重なり順を高くする
bind ID [イベント [コマンド]]	バインディングの設定

ウィジェットと同様に，図形に対してもバインディングを設定することができます。これはタグと一緒に詳しく説明します。

タグとバインド

キャンバスで作成した図形にはバインディングを設定することができます。簡単な例題として，作成した矩形をドラッグで移動させてみましょう。次のプログラムを実行してください。

最初に，一辺の長さが10の矩形を作ります。次に，その矩形に対してバインディングを設定します。イベント<B1-Motion>は，左ボタンを押した状態でマウスを動かした場合，つまりドラッグに対応します。イベントハンドラの中では，%x，%yはマウスの座標に変換されることに注意してください。新しい座標を計算してから，図形の位置をcoordsで変更します。

●タグの設定

それでは操作する矩形を3つに増やしてみましょう。それぞれの矩形にバインディングを設定してもいいのですが，同じようなプログラムをいくつも書くのは面倒ですね。このような場合，「タグ」を設定すると簡単にプログラムを記述することができます。タグ（tag）には荷札という意味があり，図形に識別子をつける働きをします。そして，図形を操作するコマンドは，操作対象となる図形の指定を，番号のほかにもタグを使って行うことができるのです。タグの設定は，図形を生成するときにオプション－tagsで行います。それではタグを指定して矩形を3つ作ります。

```
.c0 create rectangle 10 10 20 20 －fill brown －tags brown
.c0 create rectangle 20 10 30 20 －fill brown －tags brown
.c0 create rectangle 30 10 40 20 －fill brown －tags brown
```

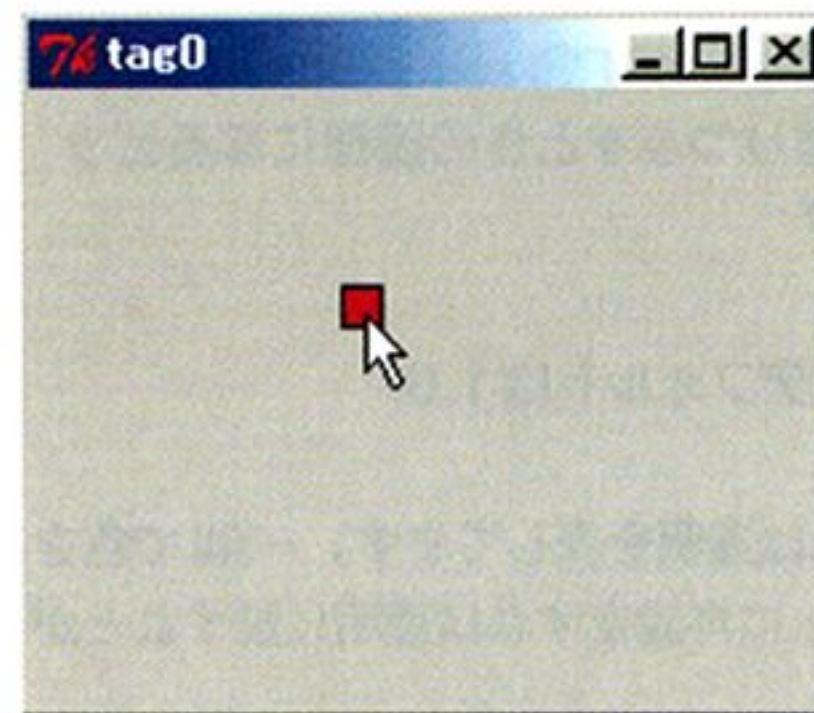



図13　矩形をドラッグして好きな位置へ

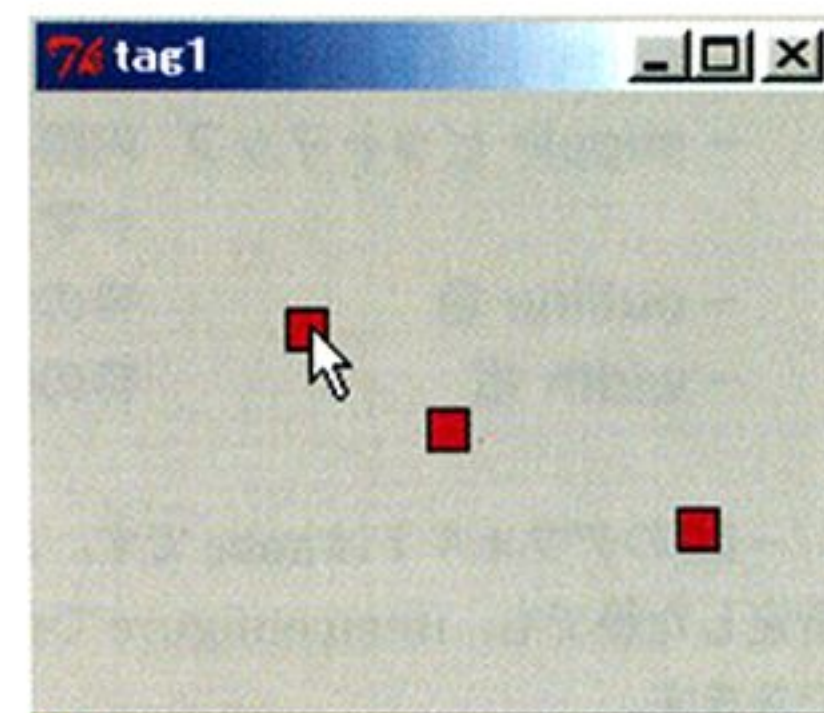



図14　タグをバインドして3個に増やしたところ

リスト14　ドラッグ

```
canvas .c0 -width 200 -height 150
pack .c0
set r [.c0 create rectangle 10 10 20 20 -fill brown]

.c0 bind $r <B1-Motion> {
   set x1 [expr %x-5]
   set x2 [expr %x+5]
   set y1 [expr %y-5]
   set y2 [expr %y+5]
   .c0 coords $r $x1 $y1 $x2 $y2
}
```

タグは文字列で指定します。今回はbrownとしました。このタグに対してバインディングを設定します。

```
.c0 bind brown < B1 － Motion > {
   set x1 [expr %x－5]
   set x2 [expr %x+5]
   set y1 [expr %y－5]
   set y2 [expr %y+5]
   .c0 coords current $x1 $y1 $x2 $y2
}
```

図形の番号ではなくタグbrownを指定します。ただし，このままでは操作対象となる矩形がわかりません。この場合，特別なタグcurrentを使います。currentはTcl/Tkが設定するタグで，マウスカーソルがある図形上にくると，その図形にタグcurrentを設定し，その図形からマウスカーソルが出るとタグcurrentを削除します。つまり，マウスカーソルが指している図形はタグcurrentで指定することができるのです。これで，複数の矩形をひとつのイベントハンドラで操作することができます。

このほかにも，タグには図形をまとめて操作することができる，という利点があります。たとえば，矩形の色をまとめて変更する場合は，タグを使って行えばいいのです。

```
.c0 itemconfigure brown －fill green
```

これでbrownの矩形は色をgreenに変更することができます。削除する場合もタグを使えば簡単です。

```
.c0 delete brown
```

これでタグ brown を持つ矩形をすべて削除することができます。

モグラたたき

それでは，最後にゲームを作ってみましょう。最初はパズルを考えていたのですが，簡単なアクションゲームならばTcl/Tkだけでもなんとかなりそうなので，「モグラたたき」を作ることにしました。5行5列の穴からモグラが出てくるので，それをマウスでクリックしてください。私は絵心のないプログラマなのでモグラの代わりに楕円を使いますが，グラフィック表示するように改造するとゲームらしくなるでしょう。

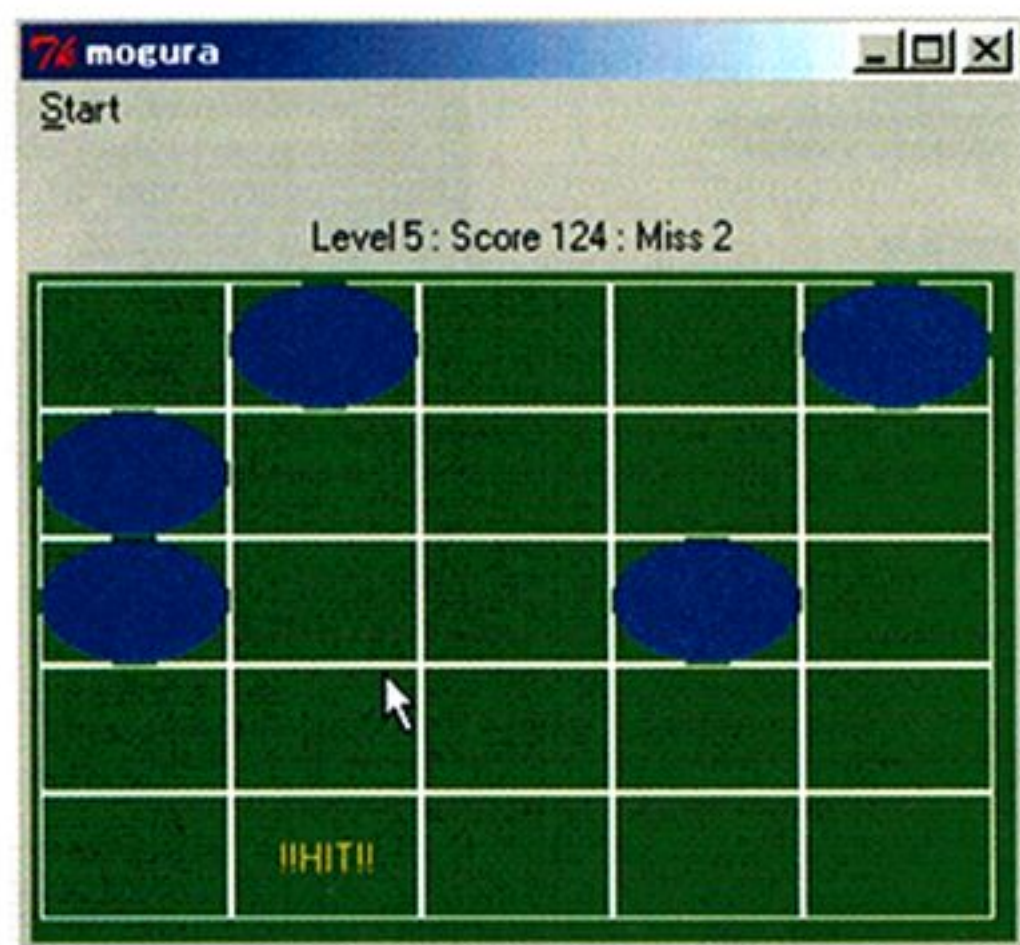



図15 モグラたたきゲーム

ゲーム作成のポイントは，モグラの出し入れとモグラを叩く処理です。モグラを叩くことはマウスのクリックで行いますので，モグラ（楕円）に対してバインディングを設定すれば簡単です。

●afterコマンド

モグラの出し入れですが，ユーザーからの入力がなくてもゲームを進行させなくてはいけないので，単純なイベント駆動型アプリケーションでは「モグラたたき」を実現することはできません。このため，プログラム自身でなんらかのきっかけを作ってやる必要があります。このような場合，役に立つコマンドがafterです。afterの機能を示します。

after msec	指定された時間（単位は msec）だけ待つ
after msec command	指定した時間経過後コマンドを実行 固有番号を返す
after cancel 固有番号	固有番号のコマンド実行待ちを取り消す
after cancel コマンド	コマンドの実行待ちを取り消す
after info [固有番号]	固有番号の実行待ちコマンド情報を返す

このように，afterは単純な時間待ちを行うほかに，一定時間後に指定したコマンドを起動するタイマーの働きも持っています。たとえば，ゲームを進行させるプロシージャをgameとしましょう。gameを一定間隔で実行させる場合，gameの最後でafterを使って自分自身の起動を設定すればいいのです。具体的には次のようにプログラムします。

```
proc game {} {
        # gameの処理
        ……
        after 500 game
}
```

これで500msec後にgameが実行されます。もっとも，厳密に500msecごとにgameが実行されるわけではありません。gameの処理にも時間がかかりますし，Windowsはマルチタスクで動作しているので，ほかのタスクの影響も受けるからです。まあ，モグラたたきのようなゲームの場合，厳密なリアルタイム処理は必要としないので，これで十分です。

gameは一定間隔で時を刻むので，これを使ってモグラを管理します。モグラたたきの場合，モグラが隠れている状態から，モグラが出る，モグラを叩く，または叩き損ねてモグラが逃げる，そしてモグラが隠れている状態に戻ります。この場合，モグラが穴を移動するのではなく，各穴にモグラが一匹ずついて出たり隠れたりする，と考えたほうが簡単です。つまり，モグラを中心に考えるのではなく，穴の状態を基準にプログラミングするのです。

モグラを出す場合は，モグラが隠れている穴の中からランダムに選べばいいでしょう。このあとの状態は，それぞれ持続時間が異なります。モグラが出ている時間を3秒，叩かれた状態を1秒間，逃げた状態を2秒間表示するとしましょう。この時間管理をgameを使って行います。具体的には，穴の状態を配列statで表し，持続時間を配列mogutimeで管理します。

state
　0：モグラが隠れている状態
　1：モグラが出ている状態
　2：モグラを叩いた状態（HIT! と表示する）
　3：モグラが逃げた状態（MISS と表示する）
　4：準備中

モグラの状態が変化するときは，stateの値を書き換えると同時に，mogutimeの値をセットします。gameはstateが0以外の場合，mogutimeの値を－1していき，mogutimeが0になったら（タイムアップ），次の状態へ変化させます。たとえば，モグラを出すときは，stateが0の穴からランダムに選び，stateを1に書き換え，mogutimeに6をセットします。gameは500msec間隔で実行されるので，これで約3秒間モグラを表示することになります。

stateが1でタイムアップした場合は，モグラを叩き損ねた場合です。このときは，stateを3に書き換えてmogutimeを4にセットします。モグラを叩いた場合は，stateを2に書き換え，mogutimeを2にセットします。この処理はモグラにバインディングされたイベントハンドラで行い，マウスの左クリックで実行されます。この処理はgameの実行とは関係なく発生します。つまり，モグラを叩く処理とgameの処理は非同期で行われるのです。

一般に，非同期の処理をプログラミングすることはたいへん難しいのですが，GUIアプリケーションの場合，イベントが非同期で発生することは当たり前のことです。Tck/Tkではイベントハンドラをプログラムするだけで済みます。

stateが2や3の状態でタイムアップしたら，stateを4の状態（準備中）にします。これは同じ穴からモグラがすぐに出てこないようにするためです。この時間は3秒間とし，タイムアップしたらstateを0に戻します。

それから，ゲームに変化をつけるためにレベルを導入します。最初はレベル1で，10匹モグラを叩くたびにレベルアップします。1匹叩くたびにレベルだけ得点が入ります。レベル1のときは1点しか入りませんが，レベルが10になると1匹叩くたびに10点入ります。そして，レベルアップするたびに，同時に出現するモグラの数を増やしましょう。10匹叩き損ねたらゲームオーバーとします。

表示にはキャンバスウィジェットを使いますが，モグラを出すたびに楕円を描画していては無駄ですね。あらかじめ必要な図形を用意しておいて，穴を表す図形の背後に隠しておくことにします。

ほとんどのウィンドウシステムでは，上にある窓が下にある窓を覆い隠します。ウィジェットにも順番があり，Tcl/Tkの場合はあとから作成したウィジェットが上になります。ウィジェットの重なり順を変更するコマンドがraiseとlowerです。

```
raise ウィジェット1 [ウィジェット2]
lower ウィジェット1 [ウィジェット2]
```

ウィジェットがひとつの場合，raiseは重なり順をいちばん上にし，lowerはいちばん下にします。ウィジェットを2つ指定した場合は，ウィジェット2に対してウィジェット1を，raiseでは上に，lowerでは下にします。

キャンバスウィジェットで作成する図形も同じです。あとから作成した図形が上となり，その順番をウィジェットコマンドraiseとlower で変更することができます。このゲームでは，穴を矩形，モグラを楕円，HIT!とMISSをテキストで描画しておき，状態にあわせてその図形の優先順位をいちばん上にします。

作成した図形のIDは配列に格納しておきます。

hole：穴の ID を格納する
mogu：モグラの ID を格納する
miss：MISS の ID を格納する
hit：HIT! の ID を格納する

穴は5行5列なので，各配列は2次元配列としてアクセスします。といっ

ても，Tclの場合は連想配列しかありません。配列を表す()の中では変数展開が行われることを使って，擬似的に2次元配列を実現します。つまり，(x, y)の位置にある穴のIDにはhole($x, $y)とアクセスします。また，その状態を求めるときはstate($x, $y)とします。

●バインディングの設定

モグラを作成したときは，バインディングを設定しなくてはいけません。これは次のように行います。

```
set mogu ($x,$y) [.c0 create oval $x1 $y1 $x2 $y2 ¥
                              -outline darkgreen -fill
blue]
.c0 bind $mogu ($x,$y) <Button-1> "attack $x $y"
```

これは図形の初期化ルーチンから抜き出したものです。作成した図形はmoguに格納し，その図形に対してイベントハンドラをバインディングしています。イベントハンドラはattackですが，引数の$xと$yは変数展開が行われるので，それぞれのモグラにバインディングされるattackには，異なる引数が与えられることに注意してください。これでクリックされたモグラを特定することができるので，わざわざマウスの位置を求める必要はありません。

●attack

イベントハンドラattackは次のようになります。

```
proc attack {x y} {
    global state mogutime mogu hit hit_count now_mogu
                                        score level
    incr hit_count
    if {$hit_count && [expr $hit_count % 10] == 0} {
        incr level
    }
    incr score $level
    incr now_mogu -1
    set state ($x,$y) 2
    set mogutime ($x,$y) 2
    .c0 lower $mogu ($x,$y)
    .c0 raise $hit ($x,$y)
    print_label
}
```

変数hit_countは叩いたモグラの数，now_moguは穴から出ているモグラの総数を表します。これらの変数はレベル管理に使います。モグラを隠している状態では，クリックしてもイベントハンドラは起動されないので，stateのチェックを行う必要がありません。このへんはwishが面倒を見てくれるので，プログラミングは簡単になります。print_labelはスコアを計算してラベルに表示します。

●game

ゲームを進めるために一定間隔で実行されるgameは次のようになります。

```
proc game {} {
    global state mogutime
    for {set x 0} {$x < 5} {incr x} {
        for {set y 0} {$y < 5} {incr y} {
            if {$state ($x,$y) != 0 } {
                incr mogutime ($x,$y) -1
                if {$mogutime ($x,$y) == 0} {
                    if [change_mogura $x $y] {
                        game_over
                        return
                    }
                }
            } else {
                gen_mogura $x $y
            }
        }
    }
    update
    after 500 game
}
```

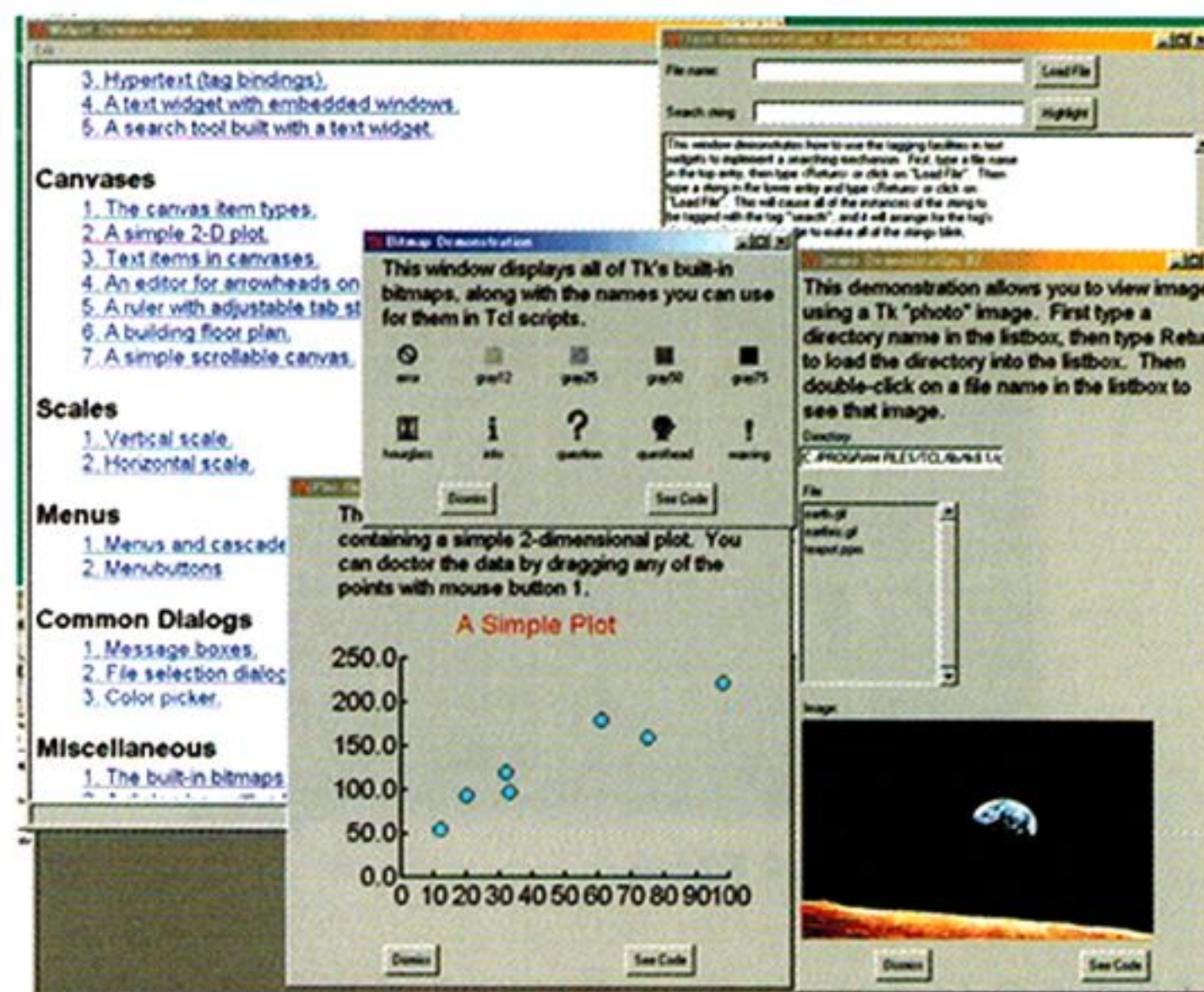



図16 WidietTourではさまざまなサンプルが見られる

gameではstateが0以外の穴についてmogutimeを−1していき，タイムアップしたならばモグラのステートを変更するためchange_moguraを呼び出します。change_moguraではモグラの状態を変更します。もしも，ミスが10回に達したら1を返すので，ゲームの終了処理をgame_overで行います。このあとはreturnでgameを抜けます。

afterの前のupdateは，待ちイベントを実行するコマンドです。モグラの出し入れやスコア表示など，Tcl/Tkでは簡単に行っているように見えますが，そのたびに画面更新のイベントが発生します。これらの処理はwishが面倒をみてくれるのですが，プログラムの実行に時間がかかると，その処理を行うことができません。このため，すべての待ちイベントを処理するupdateを実行して，画面の状態を確実に更新するようにしています。

あとは特に難しい処理を行っていないので，ここまでの説明でソースファイルは読むことができるはずです。ゲームは単純なのですぐに飽きるでしょう。いろいろと改造してみてください。

おわりに

入門編ということで，簡単で面白そうな機能をひととおり説明しました。この記事を読んでTcl/Tkに興味を持たれた方は，ぜひプログラムを作ってみてください。面白いプログラムができたらOh! Xに投稿しましょう。

このほかにも，便利で役に立つ機能はたくさんあります。特にテキストウィジェットは，テキスト編集だけではなく，マークやタグを設定することで，高度な処理が可能です。ハイパーテキストも実現できますし，PerlとTkを組み合わせた処理系Perl/Tkを使ってWebブラウザを作った人もいるそうです。Tcl/Tkでも，まだまだいろいろなことができそうなので，応用編で紹介したいと思います。

参考文献

[1] 久野靖，『入門 tcl/tk』，アスキー出版局，1997
[2] John K. Ousterhout 著，西中芳幸 石曽根信 訳，『Tcl&Tkツールキット』，ソフトバンク，1995
[3] Sriram Srinivasan 著,須田隆久 訳，『実用Perlプログラミング』，オライリー・ジャパン，1998

Tcl/Tkによるミニミニゲーム集

広井 誠　Hiroi Makoto

Tcl/Tkの実践編です。私がTcl/Tkの勉強を兼ねて作ったのが，ミニミニゲーム集です。今回は4つのゲームとさらにC言語とのリンクを行ったゲームを3つ（1種は同じ）を作りました。単体で使うだけでなく，C言語と組み合わせて，新しい可能性も模索してみます。

1 マスターマインド

0から9までの異なった4桁の数字を当てるゲームです。数字はあっているが桁は間違っている個数をcowsで表し，数字も桁もあっている個数をbullsで表します。つまり，bullsが4になると正解となります。このゲームでは，コンピュータがあなたのお相手をいたします。少ない回数で正解したほうが「勝ち」，同じ回数だと「引き分け」となります。また，両者とも10回以内に当てることができない場合も「引き分け」となります。

ファイル名
mastermind.tcl　Tclスクリプトファイル
mastermind.txt　説明書

図1
マスターマインド。半分運次第とはいうものの読み切りは重要だ

2 そろえてポン

25枚のパネルを同じ色に揃えるパズルゲームです。パネルをクリックすると，そのパネルと上下左右のパネルの色が変化します。揃える色はどの色でもかまいません。色は，2，3，4，5色の中から選ぶことができます。クリアした問題は回数とともにファイルに格納されるので，最小回数で色を揃えるようにチャレンジすることができます。

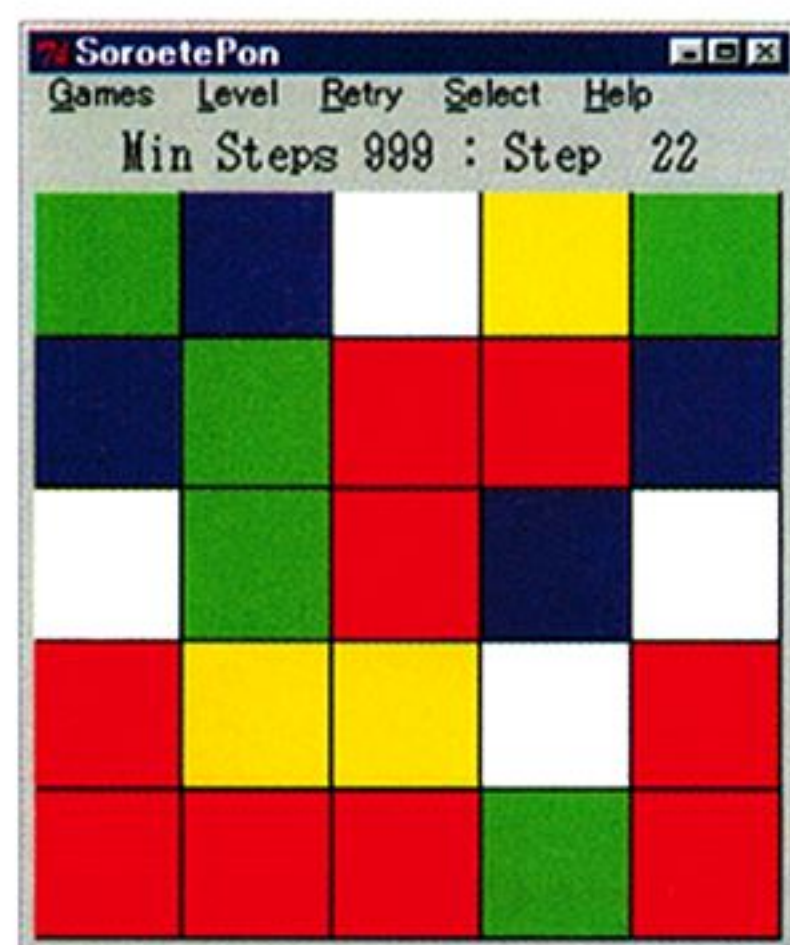



図2
色を揃えていく……はずなのに，だんだん乱れていくのはなぜ？　難しい……

ファイル名
spon.tcl　Tclスクリプトファイル
spon.txt　説明書

3 おとしてポン

SameGameやSX-BlockDownに類するパズルゲームで，上下左右に2つ以上続いている同じ色の駒を取り除いていきます。空いた場所には，上の駒が落ちてきます。縦一列の駒がすべて消えると，それより右側にある駒が左へ移動します。

同時に消した駒の数をnとすると，(n－1)の2乗だけ点数が入ります。すべての駒を消すと10万点のボーナスが入ります。また，同じ種類の駒をすべて消した場合には，1種類につき1万点のボーナスが入ります。

付属のデータには，200個の問題が用意されています。最初は簡単に高得点を狙えますが，後半になると難しくなります。スコアは保存されますので，納得するまでチャレンジしてください。

ファイル名
same.tcl　Tclスクリプトファイル
same.dat　データファイル
same.txt　説明書

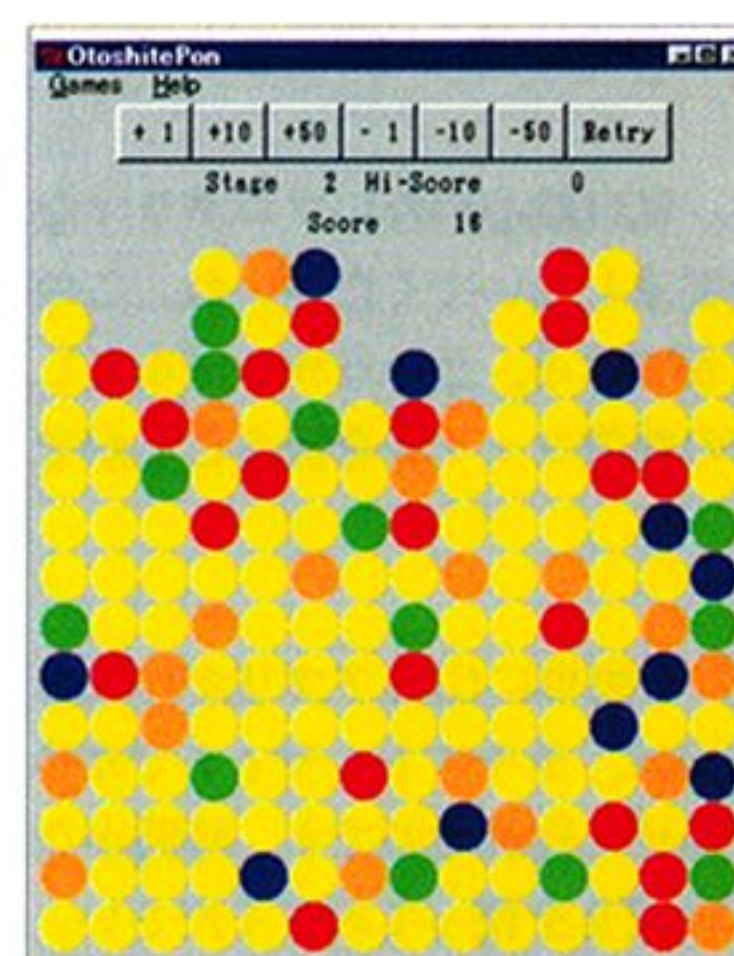



図3
いわゆる揃えてまとめて消す系統のゲームです。ハイスコアを狙え

4 カラー

相手よりもたくさんの駒を集めれば勝ち，というゲームです。ルールは単純ですが，展開がスリリングで面白い思考ゲームです。

カラーの盤面は，6個の穴が向かいあって並んでいます。左右にはカラーと呼ばれる穴があり，右隣が自分のカラーです。最初は，カラーを除く穴に6個ずつ駒が入っています。自分のカラーに半分以上の駒（36個）を集めることが勝利条件です。

ゲームは，自分の穴を選び，その中の駒をすべて取り出して，右隣の穴から反時計回りで駒をひとつずつ配っていきます。このとき，自分の穴とカラー，相手の穴には駒を入れますが，相手のカラーには入れません。駒を配り終えたときに，次のようなスペシャルケースが発生します。

① カラーで配り終えた場合，もう一度駒を配ることができる。

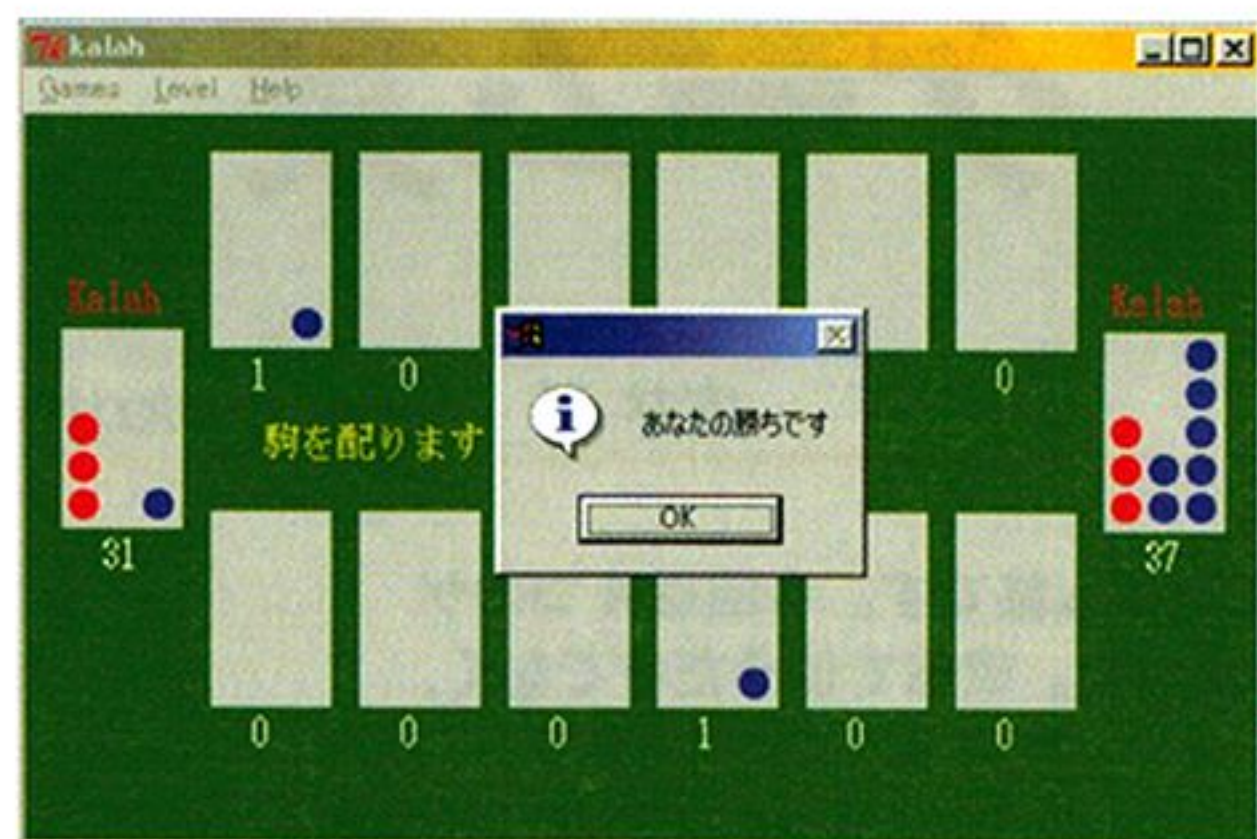



図4
わかりにくいようで，ルールを把握すれば意外とハマルかも

② 自分の空の穴で駒を配り終えた場合，向かいの穴に相手の駒があれば，両方の駒を自分のカラーに入れる。これを「両取り」という。
③ 自分の穴に駒がなくなった場合，相手の穴に残っている駒をすべて相手のカラーに入れてゲームを終了する。

駒の配置によっては，連続してカラーに駒を入れることができます。また，うまく両取りがかかるようにすると，相手の駒をいっきに自分のカラーへ入れることができます。劣勢な状態でも，相手の穴に駒がなくなれば，③の条件により大逆転も可能です。

ファイル名
kalah.tcl　　Tclスクリプトファイル
kalah.txt　　説明書

ゲームを実行するには，まずTcl/Tkをインストールしてください。私が使ったバージョンはTcl/Tk8.1a2です。Windows版はtcl812a.exeを使います。実行するとインストーラが起動され，「スタート」メニューの「プログラム」フォルダにTcl/Tkのフォルダが作成されます。

今回のゲーム集は，スコアをセーブするゲームがあるので，ハードディスクに適当なディレクトリを作って，すべてのファイルをコピーしてください。あとは，拡張子がtclのファイル(Tclスクリプトファイル)をダブルクリックすればゲームが実行されます。

どれも見たことがあるゲームだと思います。私は絵心のないプログラマなので，画面デザインについては多くのことを要求しないでくださいね。この程度のゲームならばTcl/Tkだけで作ることができる，というサンプルとして考えてください。Tcl/TkはGIF形式などの画像ファイルを扱うことができるので，不満のある方は改造しましょう。

Tcl/Tkを使ってみた感想ですが，こんなにも簡単にGUIアプリが作れるとは，正直にいってたいへん驚きました。今回は，テキストエディタでガリガリとプログラムを書きましたが，それでも簡単に作れるのです。時間がなくてプログラミングできない，という方はTcl/Tkを使ってみてください。

もっとも，Tcl/Tkに不満がないわけではありません。実は，Tcl言語にはさまざまな批判があります。Tclはシェルスクリプトを拡張したコマンド言語，いわゆる，簡易言語にすぎないので大規模なプログラム開発には向きません。今回のゲーム集でも，ソースファイルを見てもらえばわかりますが，グローバル変数を多用しているので，プログラムの見通しはあまりよくありません。また，インタプリタですから実行速度は期待できません。カラーのような思考ゲームは，Tclには不向きでしょう。

ですが，簡単にGUIを構築できる魅力は大きいですね。もともとTclはアプリケーションやツールの拡張言語として設計されました。C言語で書かれた関数をTcl/Tkにリンクして呼び出したり，逆にC言語で作ったアプリケーションからTcl/Tkを利用することもできます。つまりTclに不向きな処理は，ほかの言語で作成してリンクすればいいのです。この方法を使えば，Tcl/Tkの適用範囲も広がります。

付録CD-ROMには思考ルーチンをC言語で記述したサンプルも収録されています。その場合，Level 5で5手まで先読みしていますが，待たされることはありません。

以下では，同様にメインルーチンをCで記述してTcl/TkでGUI化したゲームを紹介します。

5 アップでポン

玉をピラミッドのように積み上げていき，先に持ち玉を使いきったほうが勝ち，という思考ゲームです。一般には「アッパーハンド」と呼ばれています。

ルールの説明

5行5列の穴が空いた盤の上に玉を積み上げていきます。このゲームでは，このほかにも7行7列，9行9列の盤もプレイすることができます。中央には，中立の玉(このゲームでは緑色の玉)を置きます。先手は赤玉を28個，後手は青玉を26個持ち，交互に玉を1個ずつ置いていきます。このとき，玉を置くことができる場所は，「穴の上」か「4個の玉が作る最小の正方形の上」のどちらかです。

玉を置くことによって最小の正方形ができたときには，次の規則を適用します。

① 正方形を構成する4つの玉のうち，3つ以上が同じ色の玉の場合は，その上に同じ色の玉を必ず置きます。同じ色の玉が2つ以下の場合は，上に玉を置きません。中立の玉は，赤と青のどちらにもカウントしません。
② 1手で最小の正方形が複数(最大4個)できたときや，①の規則で置いた玉によって新しく最小の正方形ができたときには，すべての正方形に対して①の規則を適用します。
③ すべての正方形に対して①の規則を適用したら手番を交代します。

このように，交互に玉を置いていき，先に玉を使いきったほうが勝ちとなります。それから，自分の手番のときに玉を使いきるとは限らない，ということに注意してください。自分が置いた玉によって，相手の玉がなくなる場合もあるのです。また，自分の玉を使いきり，そのことによって相手の玉がなくなることもあります。その場合は引き分けとなり，ピラミッドが完成します。

このゲームは，持ち玉をたくさん減らせるところに玉を置いていくだけでは勝てません。目先の玉数よりも，終盤でいっきに玉を積み上げることができるように，玉の配置を工夫することが重要になります。Level 1，2の思考ルーチンは玉数を基準にした単純なものなので，玉の配置に気をつけてプレイすると，簡単に勝てるようになります。

Level 3以上の思考ルーチンは，それなりに玉の配置を考慮するので，相手が有利にならないように気をつけないと勝つことはできません。Level 5以上の思考ルーチンは，森田和郎氏が考案した評価関数[*1]を参考に作成したもので，Level 3，4よりもさらに強くなります。森田氏考案の評価関数は精度が高く，これをそのままプログラムすると強すぎるので，わざと精度を落として弱くしています。それでも勝つことは至難の業でしょう。

電脳倶楽部に掲載されたX68000版では，Level 5が強すぎるとの評価をいただきましたが，世の中は広いものでLevel 6を撃破する強者もいまし

図5
アップでポン。ルールを把握するのがまず大変かも。かなり奥が深そう

た。コンピュータの強さはX68000版と同じですが，新しく9行9列版を用意しましたので，お楽しみください。

ファイル名
updepon.tcl　Tclスクリプトファイル
updepon.dll　思考ルーチン
updepon.txt　説明書

＊1　局面を点数化してどちらが有利か判定するプログラムのこと。複雑なゲームほど精度の高い評価関数の作成が困難になる。

6 リバーシ

思考ゲームの定番「リバーシ」です。ルールの説明は不要ですね。思考ルーチンは，旧Oh!Xと電脳倶楽部で連載された「C調言語PRO-68K」で，祝一平氏が作成されたプログラムを参考にしています。激光電脳倶楽部[8]に掲載されたX68000版では，高速化のために「反復深化」というテクニックを使いましたが，Windows版ではCPUが高速なので，「ミニマックス＋αβ枝刈」のみでプログラムしました。評価関数には改良を加えていますので，X68000版よりも多少は強くなったと思います。

序盤の定石は入っていませんが，終盤では残り9（Level 1）～14（Level 5）手になると最後まで読み切るようになっています。「終盤での大逆転！」というリバーシの醍醐味を存分に味わってください。最新機種のパソコンであれば，このプログラムのままでも残り15手を現実的な時間で読み切れると思います。思考ルーチン側は残り19手までの読み切りに対応しているので，興味のある方はTclスクリプトを書き換えてみてください。

リバーシはコンピュータ向きのゲームと考えられていて，昔からたくさんのプログラムが作られてきましたが，人間の世界チャンピオンにはなかなか勝つことはできませんでした。それが，一昨年に「ロジステロ」というプログラムが，ついに世界チャンピオンを倒しました。ロジステロは，序盤の定石をたくさん覚えていて，終盤では残り28手を読み切るそうです。リバーシの指し手の総数は60手なので，半分とちょっと打った時点で，コンピュータ側が最善手を打ってくるわけですから，人間が勝てないのはしかたがないところでしょう。それにしてもハードウェアの進歩はすごいですね。そのうち，必勝手順が解明されるかもしれません。

ファイル名
reversi.tcl　Tclスクリプトファイル
reversi.dll　思考ルーチン
reversi.txt　説明書

ゲームの実行

ゲームを実行するには，まずTcl/Tkをインストールしてください。私が使ったバージョンはTcl/Tk8.1a2です。Windows版はtcl812a.exeを使います。実行するとインストーラが起動され，「スタート」メニューの「プログラム」フォルダにTcl/Tkのフォルダが作成されます。あとは，拡張子がtclのファイル（Tclスクリプトファイル）をダブルクリックすれば，ゲームが実行されます。

図6
リバーシ。定番中の定番ゲーム。弱めのレベルもあるので普通の人も遊べます

Tcl/TkとC言語とのリンク

Tcl/TkとC言語とのリンクは，思っていたよりもずっと簡単でした。今回は，時間のかかる思考ルーチンをC言語で作成し，これをDLLとしてコンパイルしてTcl/Tkとリンクしました。DLLというとなにやら難しいイメージがありますが，DLLのロードはwishが面倒を見てくれるので，その規則に合わせてDLLを作成するだけで済みました。C言語とのリンクは須栗歩人氏の記事[3]がとても参考になります。

Tcl/TkとC言語間のデータの受け渡しは，用意されているライブラリ（TclLibrary Procedures）を使えば簡単にできます。英文ですがヘルプも完備しているので，基本を理解すれば，必要な関数を探し出すことは難しくありません。

ライブラリは，設計の善し悪しによって使い勝手が大きく変わるものです。なかには，ソースを読まないと動作がよくわからない，という代物もありますが，Tcl/Tkのライブラリはとても使いやすいものでした。作者のJohn K. Ousterhout博士に感謝いたします。

おわりに

8ビット時代のマイコンでは，BASICとマシン語を組み合わせてプログラムを作成することがよく行われました。Windows全盛のこの時代では，インタフェイスはTcl/Tkのようなスクリプト言語で記述し，処理速度が必要な部分はC言語のようなコンパイラで作る，というスタイルは有効なテクニックのひとつでしょう。

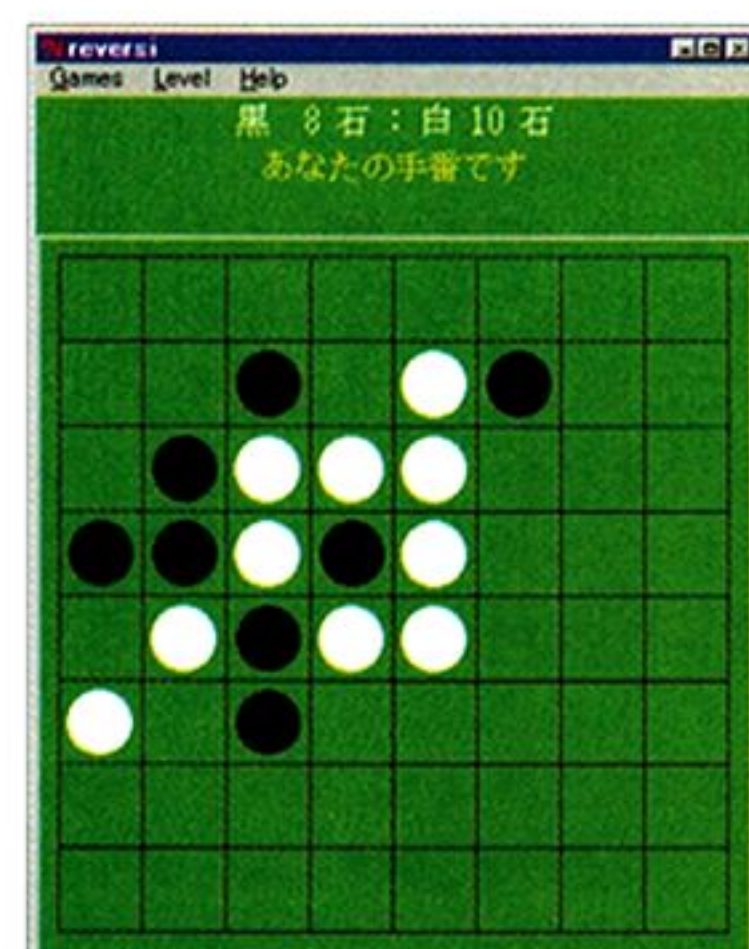
まあ，ひとつの言語で済ますことができれば，それがいちばんいいのでしょうが，GUIの作成には手軽に実行できるインタプリタが適しているように思います。そういう意味で，最近はTkとPerlを組み合わせた処理系Perl/Tkに注目しています。私の環境（Pentium/166MHz）でパズル「8クイーン」を解かせたところ，PerlはTclの6倍の速さで動作しました。さすがにPerlは速いですね。CGIはPerlで作成するのが常識になっていますが，Perl/Tkが普及すると，GUIアプリもPerlで作るのが当たり前になるかもしれません。Perlもフリーで使用できるので，Tclは性に合わない方はPerlに挑戦してみてください。Perlは個性的で面白い言語ですよ。

参考文献

●Tcl/Tk関連
[1] 久野靖，『入門 tcl/tk』，アスキー出版局，1997
[2] John K. Ousterhout，西中芳幸 石曾根信 訳，『Tcl&Tkツールキット』，ソフトバンク，1995
[3] 須栗歩人『Tcl/Tk による Windows GUI プログラミング』，C MAGAZINE 1998年11月号，ソフトバンク

●ゲーム関連
[4] 広井誠，『落としてポン！』，月刊・電脳倶楽部 1996年8月号（Vol.99），満開製作所
[5] 広井誠，『KALAH』，月刊・電脳倶楽部 1997年6月号（Vol.109），満開製作所
[6] 広井誠，『そろえてポン』，月刊・電脳倶楽部 1998年3月号（Vol.118），満開製作所
[7] 広井誠，『アップでポン』，月刊・電脳倶楽部 Vol.112，満開製作所
[8] 広井誠，『リバーシ』，激光電脳倶楽部 Vol.2，満開製作所

●マスターマインド，カラーの思考ルーチンについて
[9] 広井誠，『続・サルでも書けるCプログラム講座第20回』，月刊・電脳倶楽部 1996年12月号（Vol.103），満開製作所
[10] 広井誠，『Lisp 講座 お気楽ごくらくプログラミング入門第22回』，月刊・電脳倶楽部 1998年2月号（Vol.117），満開製作所
[11] Leon Sterling，Ehud Shapiro，松田利夫 訳，『Prologの技芸』，共立出版，1988

●思考ルーチンについて
[12] 祝一平，『C調言語PRO-68K』，月刊・電脳倶楽部 Vol.20，21，24，満開製作所
[13] 広井誠，『続・サルでも書けるCプログラム講座第20回』，月刊・電脳倶楽部 Vol.103，満開製作所
[14] Leon Sterling，Ehud Shapiro，松田利夫 訳，『Prologの技芸』，共立出版，1988
[15] 松原 仁，『将棋とコンピュータ』，共立出版 1994
[16] 松原 仁・竹内郁雄 編，『bit別冊 ゲームプログラミング』，共立出版，1997

VisualC++で始めるWindowsプログラミング（2）
MDIの基礎と画像表示

菊地 功　Kikuchi Isawo

VisualC++とMFCを使ったプログラミング入門です。前回のグラフィックエディタもどきに画像読み込み/セーブ機能をつけ，全体をMDI形式で組み直します。さらにレジストリ操作なども行ってみましょう。

前回は，冒頭で「本誌の1発目だから簡単にVC++の説明をする」という単発的な記事であることを宣言したかと思うと，最後に次号を匂わせるような表現をしてしまい，ちょっと混乱した読者もおられるのではないだろうか（筆者もいま読み返して気がついた）。なんせ行き当たりばったりで書いてるもんで。行き当たりばったりで書かせてくれる本もすごいと思うけど。で，この記事はというと，どうやら連載のような扱いになるらしい。つまり，毎号（続く限り）できるらしい。ま，行き当たりばったりだから，いつ気が変わって終わってしまうかもしれないけど。今回は，レジストリとかMDI，画像関連を予告してあったようなので（これも読み返して思い出した），その辺を重点的にやってみよう。

とはいえ，5カ月以上ブランクがあると，前回のことなんか忘れちゃうよな。ちゃんと思い出しといてね。筆者も前回の執筆から5カ月以上経ってるわけだが，結局締め切りの1カ月前にならないとなにもやり始めないんだよね。これじゃ，月刊とたいして変わらんな。量は多いけど。溜めちゃうのは筆者の性分，思い起こせば小学校の夏休みの宿題……とまあ，そんなことはどうでもいいんだが，この5カ月の間にVisualC++（Developers Studio）はVer.6.0が出てしまった。5.0とはヘルプがHTMLになって使いにくくなったくらいの違いだから，特に使い方の説明なしに6.0で行こうと思っていた。

それはクリスマスすぎの寒い夜（原稿執筆時点はそういう時期なのだ），マイメインマシンを突然襲った昨年2度目のディスククラッシュ！　普段ならそこでめげて逃避してしまうのだが，今回はそうもいかない。この記事だけは年内に上げたい（上げないとたったひとりで編集をしている（U）氏が死ぬ），というわけで，現在はサブノートで執筆中。ま，原稿を書き始める前でまだよかった（編注：実はこの頃は全然Oh!Xに取りかかれる状況ではなかった）。

しかし，このマシンにはまだ5.0しかインストールされていない。CD-ROMドライブはついてないから，インストールするにはメインマシンとLANでつながなければならない。そんなわけで，今回も5.0でいく。さっきも書いたとおり，6.0になってもなにも変わってないと思ってよい。こんなんで高いバージョンアップ料払わされたのは癪だが。

ただし，ひょっとして途中でメインマシンが復旧したら，6.0になるかもしれないが，別に5.0で作ったプロジェクトはそのまま6.0で開けるし。

画像ファイルを表示する

唐突だが，まずは画像ファイルの表示からやってみよう。VisualBasicなら，適当にピクチャーボックスに画像ファイルをパクッと喰わせてやれば表示できる（と思った）のだが，VC++だとなかなかそうもいかない。いちいちファイルを開いてメモリを確保してヘッダを解析して……なんて，うぎぃ！　面倒臭ぇ！　なわけなんだが，だからといって逃げるわけにもいかんだろう。あと，今回はMDIにしてみよう。MDIというのはMultiple Document Interfaceの略で，日本語でいうと複数文書インタフェイスである。多くのウィンドウはSDI（Single Document Interface）で，前回のサンプルもSDIであったが，Developer Studioのような大元のウィンドウの中に小さなウィンドウが複数開けるもの，これをMDIという。

まずはDeveloper Studioを立ち上げて，プロジェクトを作ろう。［ファイル］-［新規作成］メニューだ（図1　新規作成）。前回を思い出してくれよ。ここのプロジェクトタブでMFC AppWizard（exe）を選び，プロジェクト名は"GrpView"とでもしておこう。すると，AppWizardが立ち上がるので，ステップ1でMDIを指定する（図2　ステップ1）。

ステップ2，3はそのままでいいとして，ステップ4では，前回も少しだけ話に出たが，ここで［詳細設定］ボタンを押すと，扱うファイルを設定できる（図3　ステップ4，図4　詳細設定）。ここの［ファイルの拡張子］に"bmp"と指定しよう。すると，［フィルタ名］の部分も，"GrpVie ファイル（*.bmp）"となるはずだ。これが，ファイルダイアログの［ファイルの種類］で表示される文字列となる。あとは前回同様にWizardを進めれば，スケルトンは完成だ。

縁起モノなので，まずはいきなりビルド&実行してみよう。親フレームの中に，子フレームがひとつ表示されたはずだ。［新規作成］をすれば，子フレームがぽこぽこといくつでも開ける（図5　GrpView-1）。これがMDIだ。

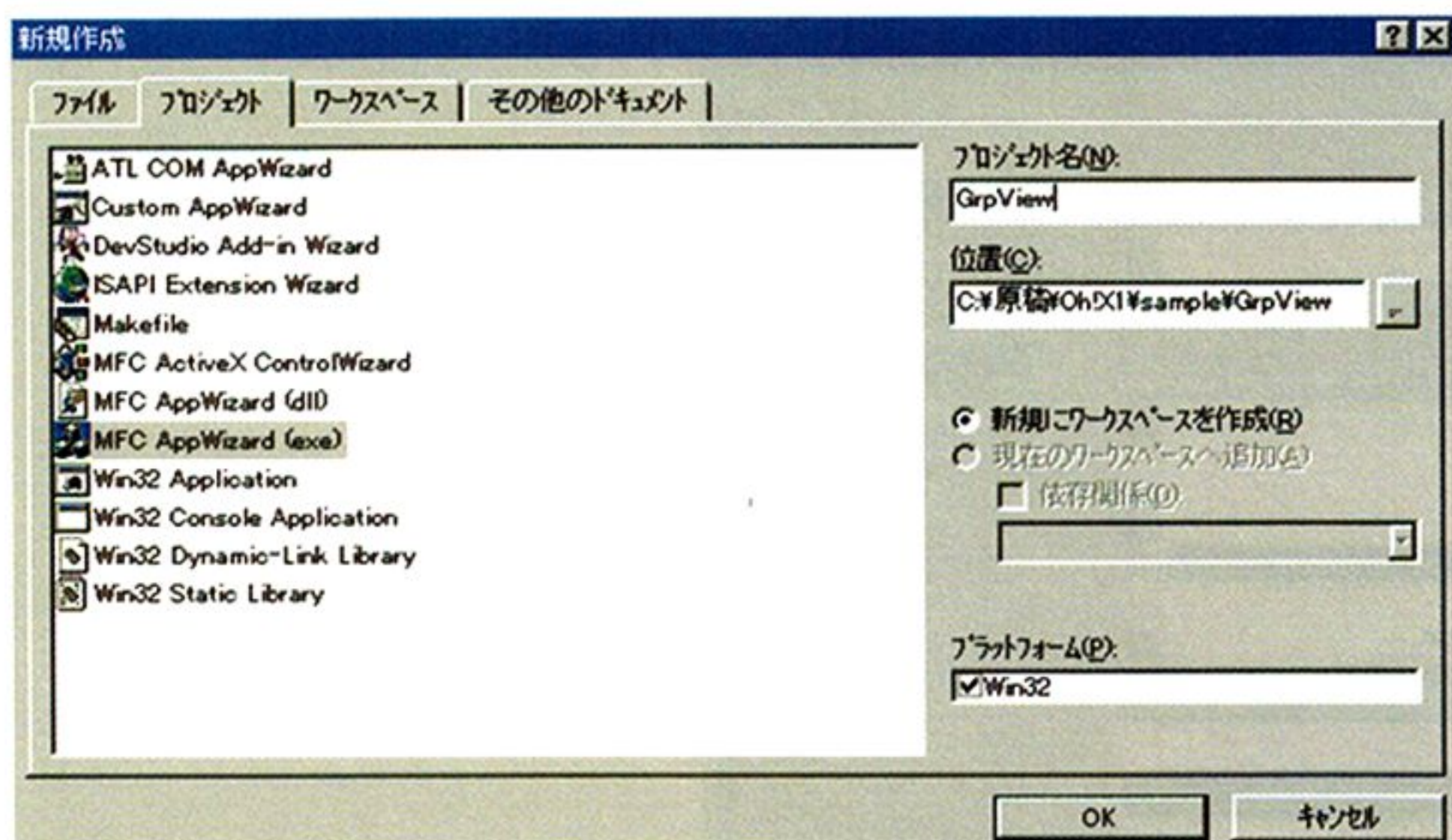



図1　新規作成

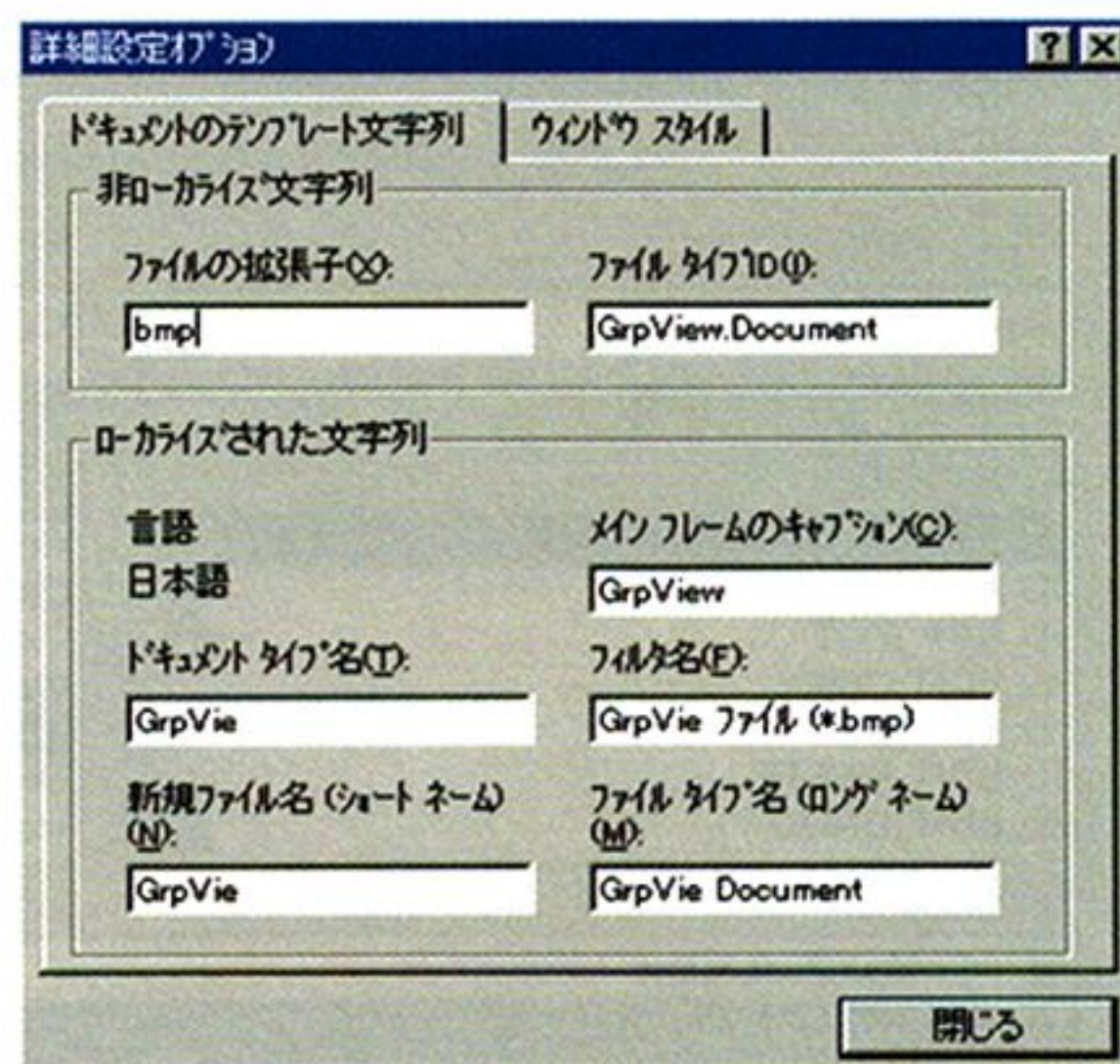



図4　詳細設定

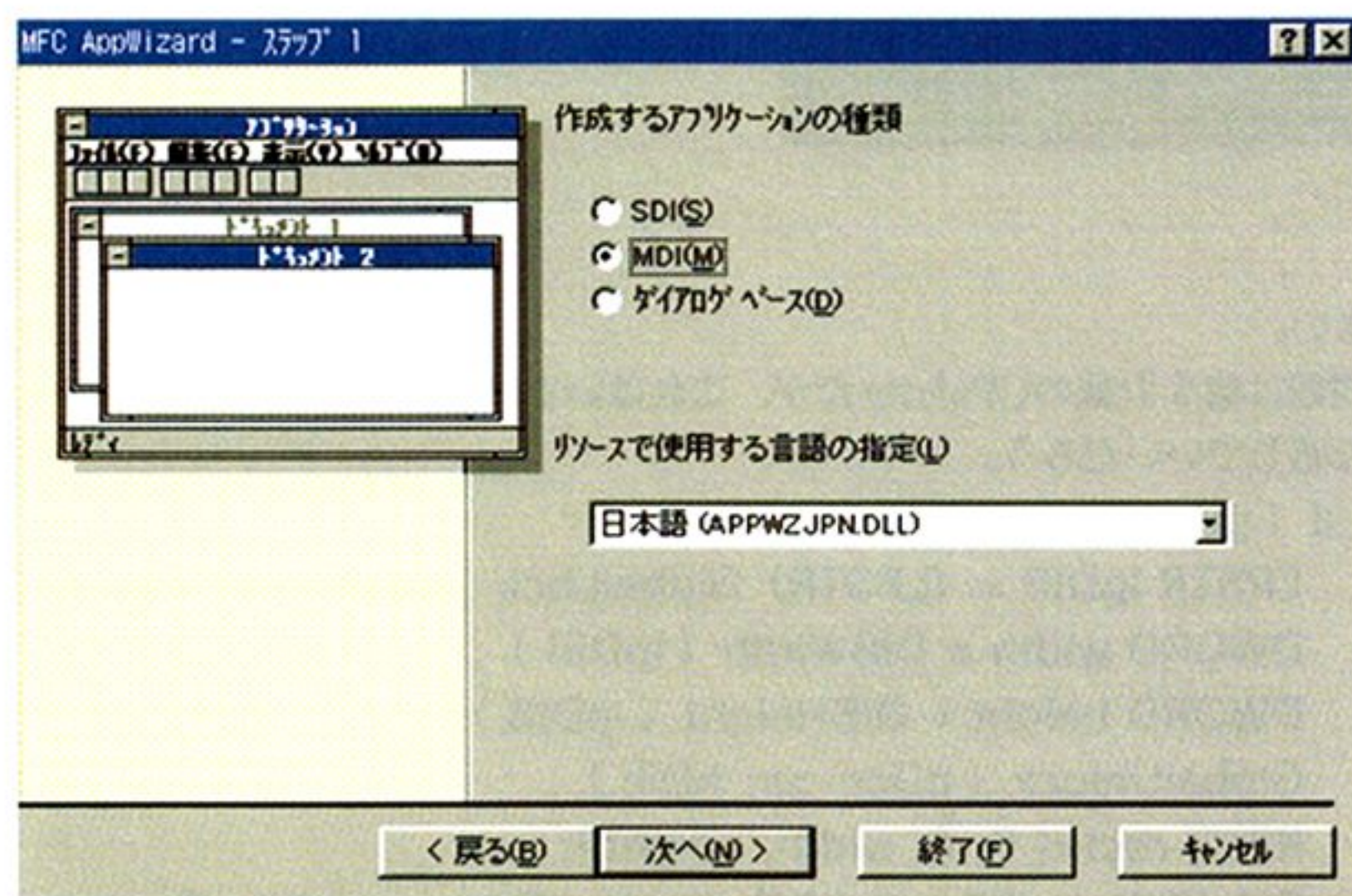



図2　ステップ1

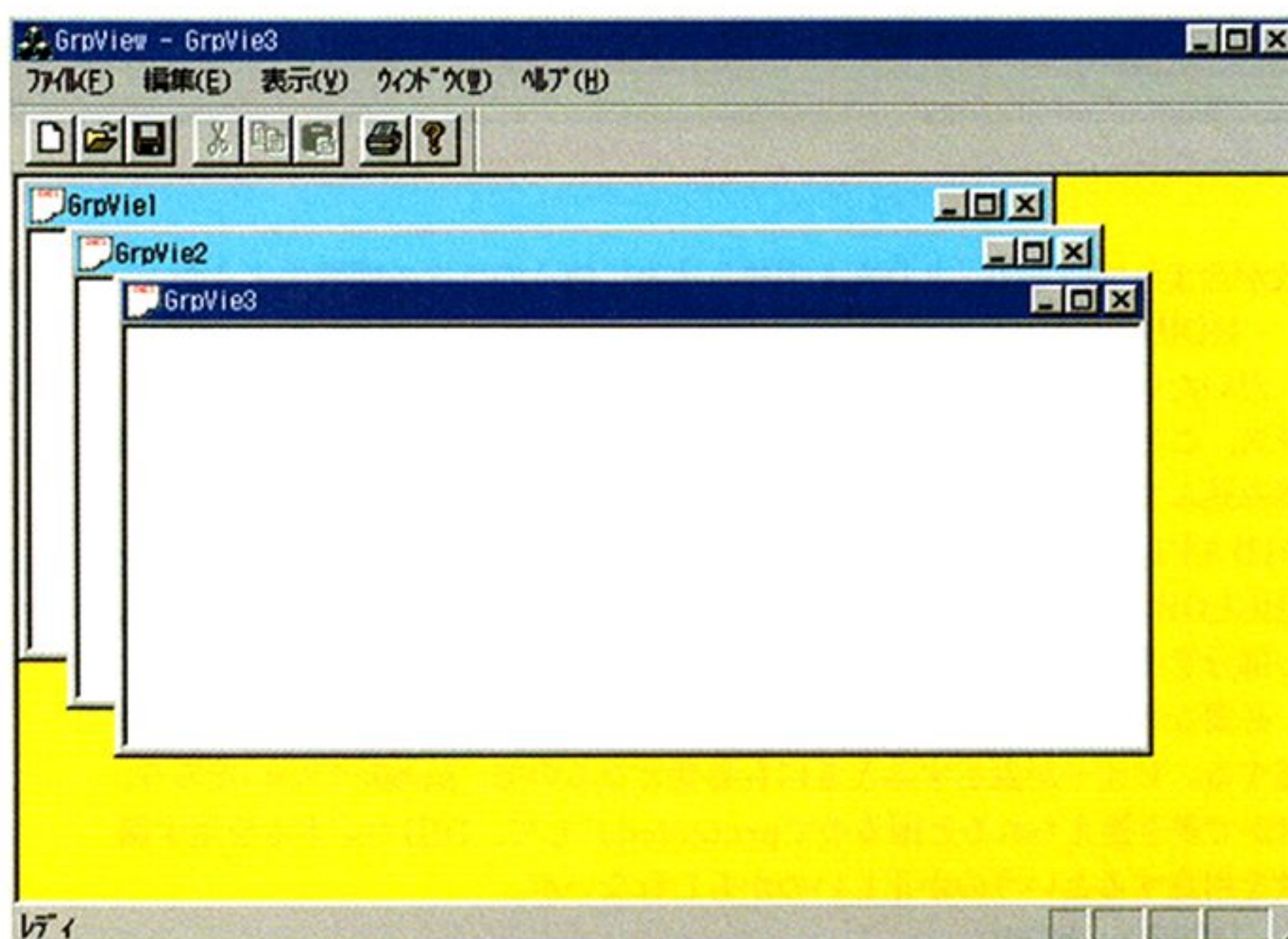



図5　GrpView-1

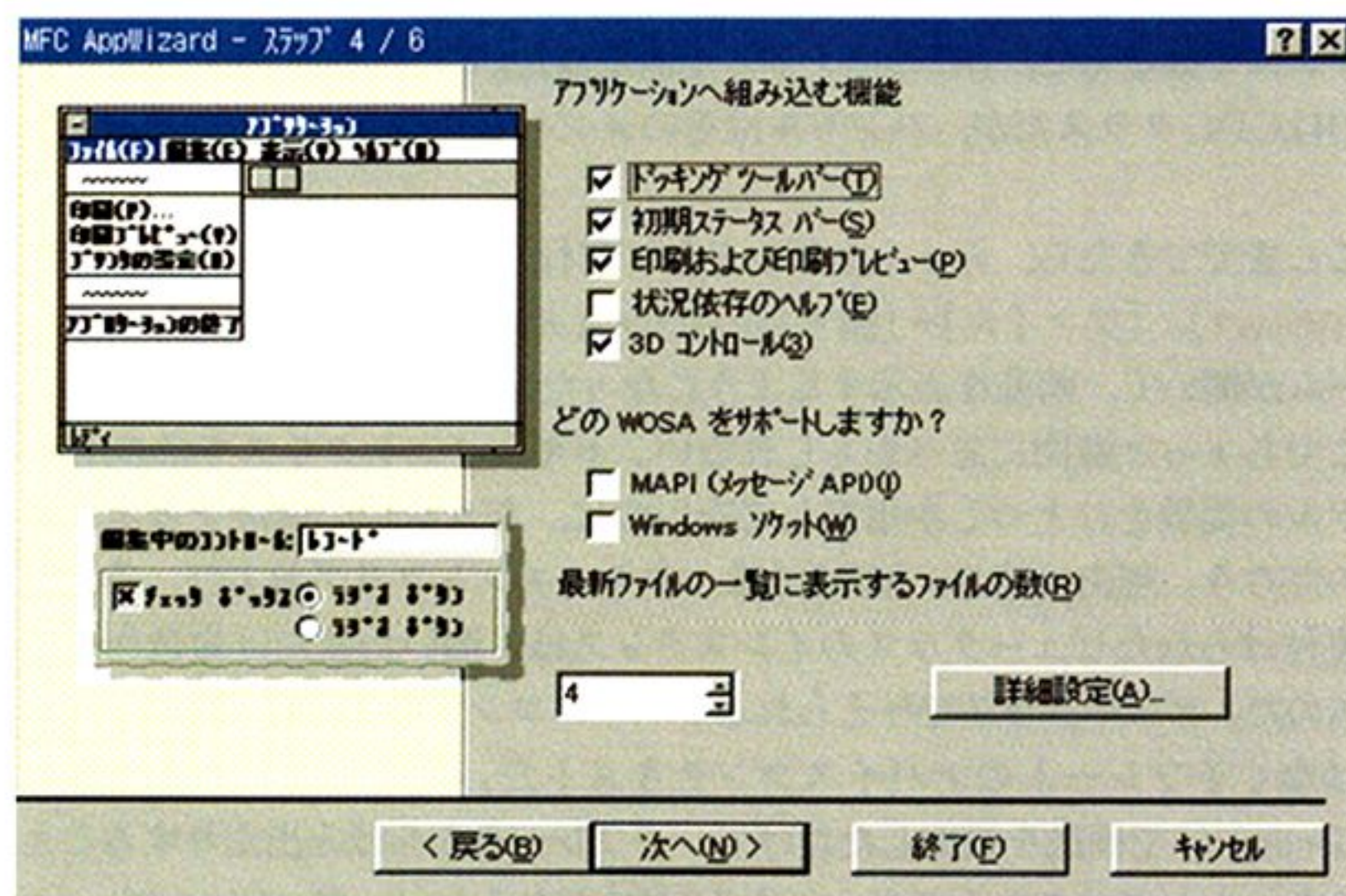



図3　ステップ4

図6　開く

今度は[ファイル]-[開く]でファイルダイアログを開いてみよう。ファイルの種類に先ほど指定した"GrpVieファイル(*.bmp)"と表示されているのがわかるだろう(図6　開く)。この時点で，すでにBMPファイル(とフォルダ)だけしかファイルリストに表示されなくなっている。ではいったんGrpViewを終了して，ソースを書き込んでいこう。

先ほど，画像ファイルを表示するのは面倒臭いと書いたが，実は単に表示するだけならばそれほどでもない。実は，VC++にはBMPファイルを表示するサンプルソースが含まれていて，そこからファイルを読み込む部分を引っ張ってくればいいだけだ(ひょっとするとLearning Editionにはついていないかも)。サンプルのプロジェクト名はDIBLOOK，Developer StudioのヘルプからDIBLOOKを検索すれば，そのページからプロジェクトをHDDにコピーすることができる。もし必要ならば，これもビルド&実行してみて，動作を確認してもいいだろう。

このサンプルのうち，BMPファイルの読み込みに必要なのはDIBAPI.CPPとDIBAPI.H，それにMYFILE.CPPだ。まずはこの3つのファイルをGrpViewプロジェクトのフォルダにコピーし，プロジェクトに加えよう。[プロジェクト]-[プロジェクトへ追加]-[ファイル]を選択し，3つのファイルを指定すればよい。まずはDIBAPI.Hを開いてみよう。いくつかの関

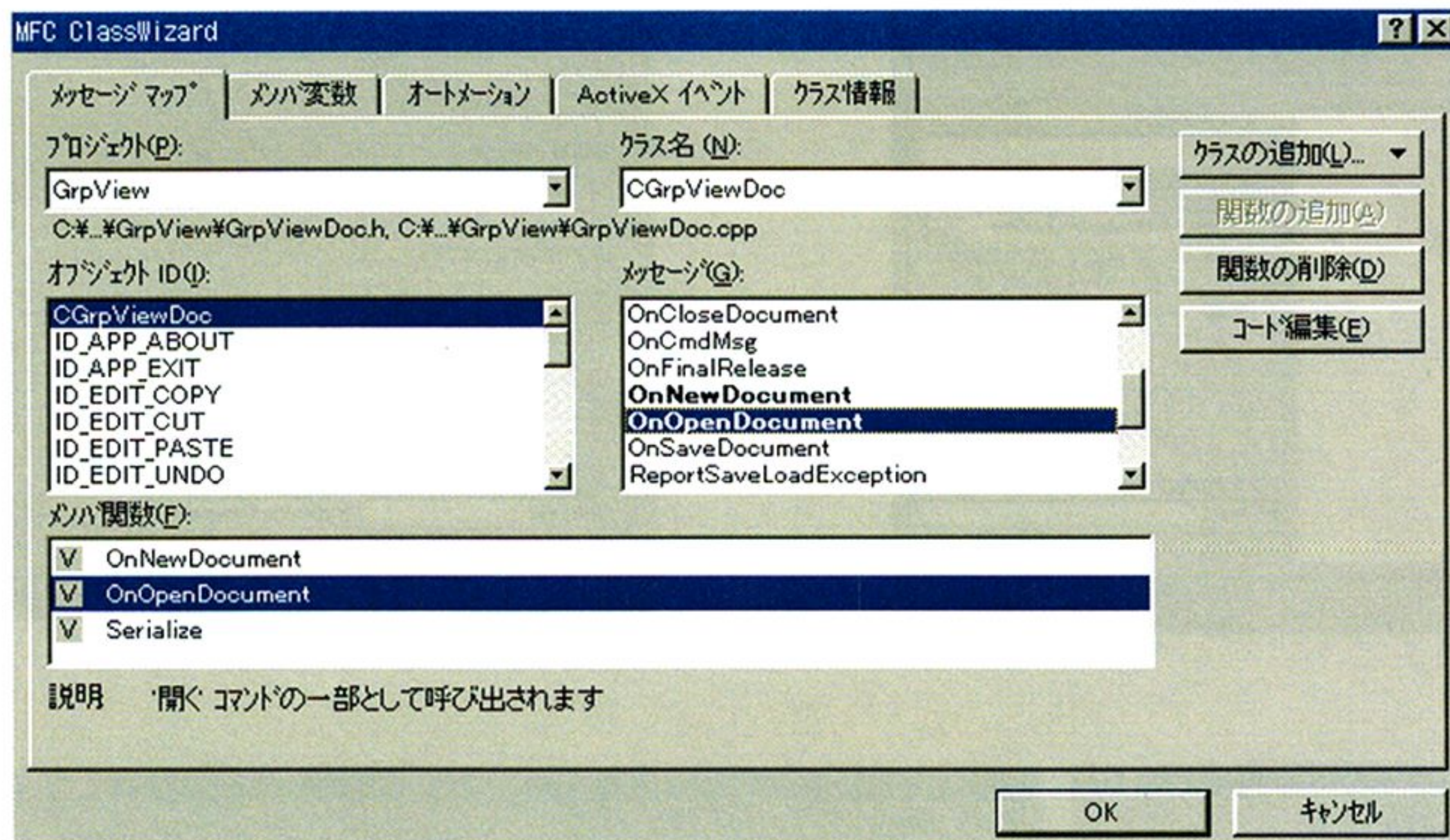



図7　classwizard

数が含まれているが，とりあえず読み込みに使うのは次の関数のようだ。

```
HDIB WINAPI ReadDIBFile（CFile& file）;
```

だいたい予想はつくだろう。CFileというのはファイルを扱うMFCのクラス，このクラスのインスタンスでBMPファイルを指定すれば，メモリに読み込んでHDIB，つまりDIBハンドルにして返してくれる。このHDIBもDIBAPI.H内で定義されているが，要はDIB専用のメモリハンドル（HGLOBAL）だと思えばよい。では，まずはドキュメントクラスに読み込む部分を書いてみよう。

必要なのは，DIBハンドルを保存する変数。これをGrpViewDoc.hに記述する。ビューが表示するときにも必要となるので，publicでいいだろう。ほかで書き換えられると困るのでprotectedにして，DIBハンドルを返す関数を用意するというのが正しいのかもしれないが。

```
HDIB m_hDib;
```

とでもしておこう。ただし，HDIBは先ほどもいったようにDIBAPI.Hで宣言されているので，それをインクルードしておかなければならない。

今度はCGrpViewDoc.cppのほうだ。まずはコンストラクタでm_hDibの初期化と，デストラクタで廃棄をしてから，ファイルが開かれたときのメッセージ関数として，ClassWizardでOnOpenDocument（）メソッドを追加する（図7　classwizard）。

このメソッドは，引数としてファイルのフルパスが渡されるが，先ほどのReadDIBFile（）関数はCFileクラスの引数を取るので，まずはそのインスタンスを作ろう。といっても，コンストラクタでファイル名とモードを指定するだけでいいので，非常に簡単だ。

```
CFile file（lpszPathName, CFile::modeRead）;
```

これだけだ。あとは，

```
m_hDib = ReadDIBFile（file）;
```

とするだけで，m_hDibに指定したファイルのDIBハンドルが入る。

では今度は表示。もちろんビュークラスGrpViewView.cppだ。画面表示はOnDraw（）メソッドでやるのは覚えているかな。ここで，今度はDIBAPI.H内だ。

```
BOOL WINAPI PaintDIB（HDC, LPRECT, HDIB, LPRECT,
                                        CPalette * pPal）;
```

第1引数は，書き込むべきデバイスコンテキストハンドル，第2引数はその領域RECTへのポインタだ。次いで，DIBハンドルと，その領域となる。実寸表示するのであれば，領域にはともにDIBのサイズを入れればよい。サイズは，

```
DWORD WINAPI DIBWidth（LPSTR lpDIB）;
DWORD WINAPI DIBHeight（LPSTR lpDIB）;
```

で，この関数の引数はDIBハンドルをロックしたメモリポインタを入れればよい。

問題は第5引数のCPaletteだが，これはいまはNULLとしておこう。こんな感じでいいだろう。

```
if（pDoc->m_hDib）{
  LPSTR lpDIB =（LPSTR）GlobalLock（pDoc->m_hDib）;
  DWORD width = DIBWidth（lpDIB）;
  DWORD height = DIBHeight（lpDIB）;
  GlobalUnlock（pDoc->m_hDib）;
  RECT rect={ 0, 0, width, height };
  PaintDIB（pDC->m_hDC, &rect, pDoc->m_hDib, &rect,
                                          NULL）;
}
```

ここで注意が必要なのは，PaintDIB（）の第1引数はCDCクラスのインスタンスではなくて，DCのハンドルである点だ。OnDraw（）で与えられるのはCDCクラスだが，ハンドルはそのメンバm_hDCで得ることができる。

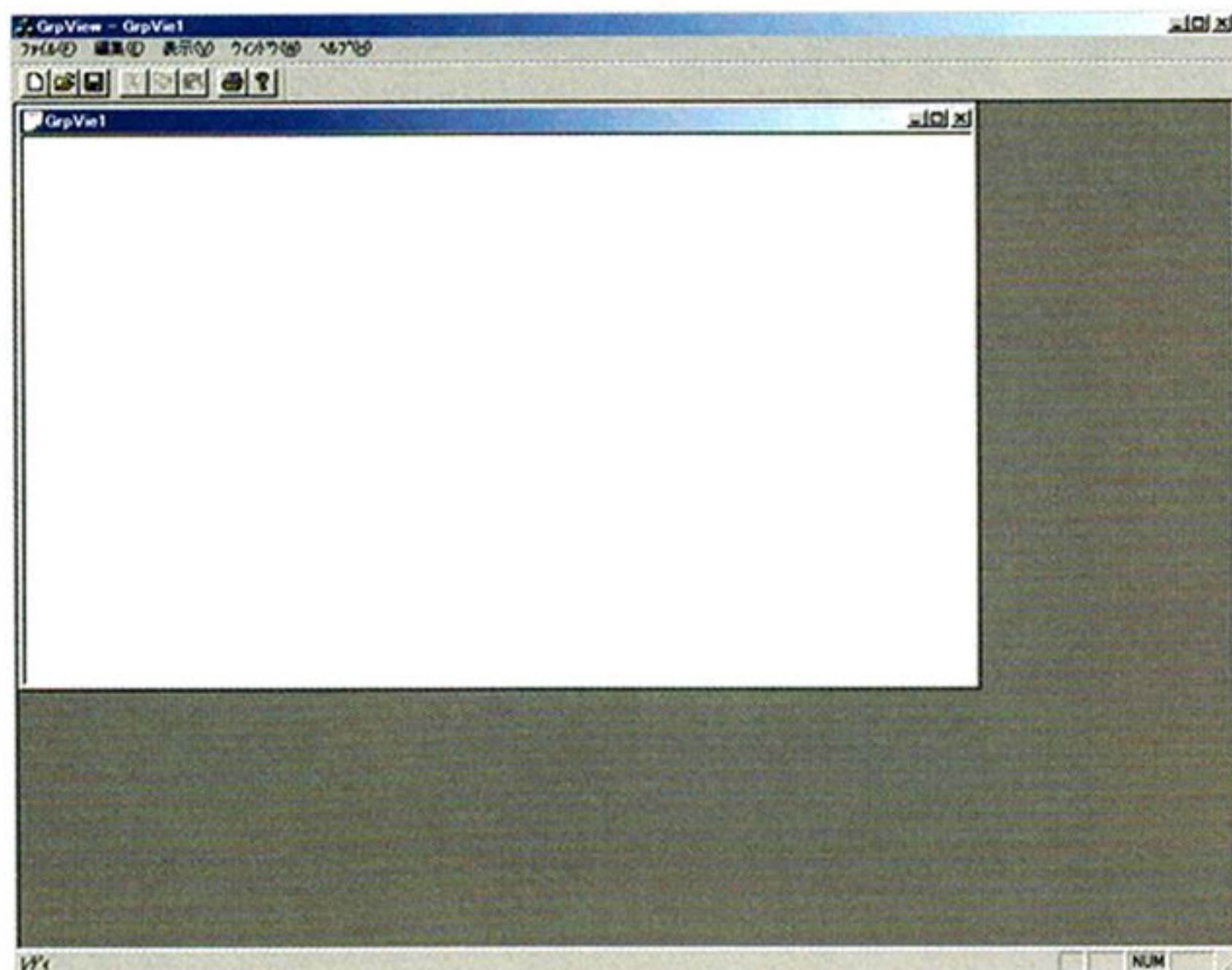
ここまでできたら，ちょっとビルドして実行してみよう（図8　サンプルGrpView1）。［ファイル］－［開く］からファイルを指定すると，新しい子フレームが開いて，画像を表示するようになったはずだ（図9　GrpView-2）。ここでちょっと疑問に思うかもしれない。ドキュメントクラスではDIBハンドルの変数をひとつしか用意してないのに，なぜ複数の画像を表示できるのだろう。理由は簡単。MDIでは，ドキュメントクラスおよび，それと関連付けられたビュークラスのインスタンスは，開いた数だけ複数作成されるのだ。ビュークラスで与えられるデバイスコンテキストも，親フレームではなく子フレームのデバイスコンテキストだ。だからなにも考えずにOnDraw（）で画像を描画しただけで，子フレームからはみ出たりすることなくちゃんと表示されるのだ。つまりMDIだからといって，ClassWizardでMDIを選択してしまえば，MFCを使っている限りそれほど特殊な操作は必要ないわけだ。

子フレームの調整

さてこの状態で，問題点を挙げてみよう。まずは，起動時点で真っ白な子フレームがひとつ勝手に開かれてしまう。これはスケルトンに組み込まれたソースが，起動時に［新規作成］と同様の操作をしてしまうからだ。アプリケーションの性質上，新規作成も不要なのだが，まずは最初に開くウィンドウをなんとかしよう。それと，画像を開いたとき，その子フレームよりも画像のほうが大きくても，スクロールバーが表示されない。これも大きな問題だ。

図8　サンプルGrpView1

図9　GrpView-2

図10

起動時の新規作成は，CGrpViewクラスのInitInstance（）中，ProcessShellCommand（）内で行われる。このメソッドは，起動時のオプションを解析し，ファイル名が指定されていればそれを開くなどといったことを行うものだ。この関数に渡されているCCommandLineInfoクラスのインスタンスcmdInfoが，もし新規作成しなさい，というものならば（つまりGrpView.exeのオプションにファイルが指定されていなかったとき），このメソッドをスキップさせてしまえばよい。というわけで，次の1行をその前に加える。

```
if ( cmdInfo.m_nShellCommand! =
                    CCommandLineInfo::FileNew )
```

こうしておけば，もしオプションでファイル名が指定されれば，そのファイルを開くが，指定されなかった場合はなにもアクションを起こさなくなる。ちょっと泥臭いが，MFCがやっていることにケチをつけようと思ったら，少々面倒なことになるというのは前回述べたとおりだ。

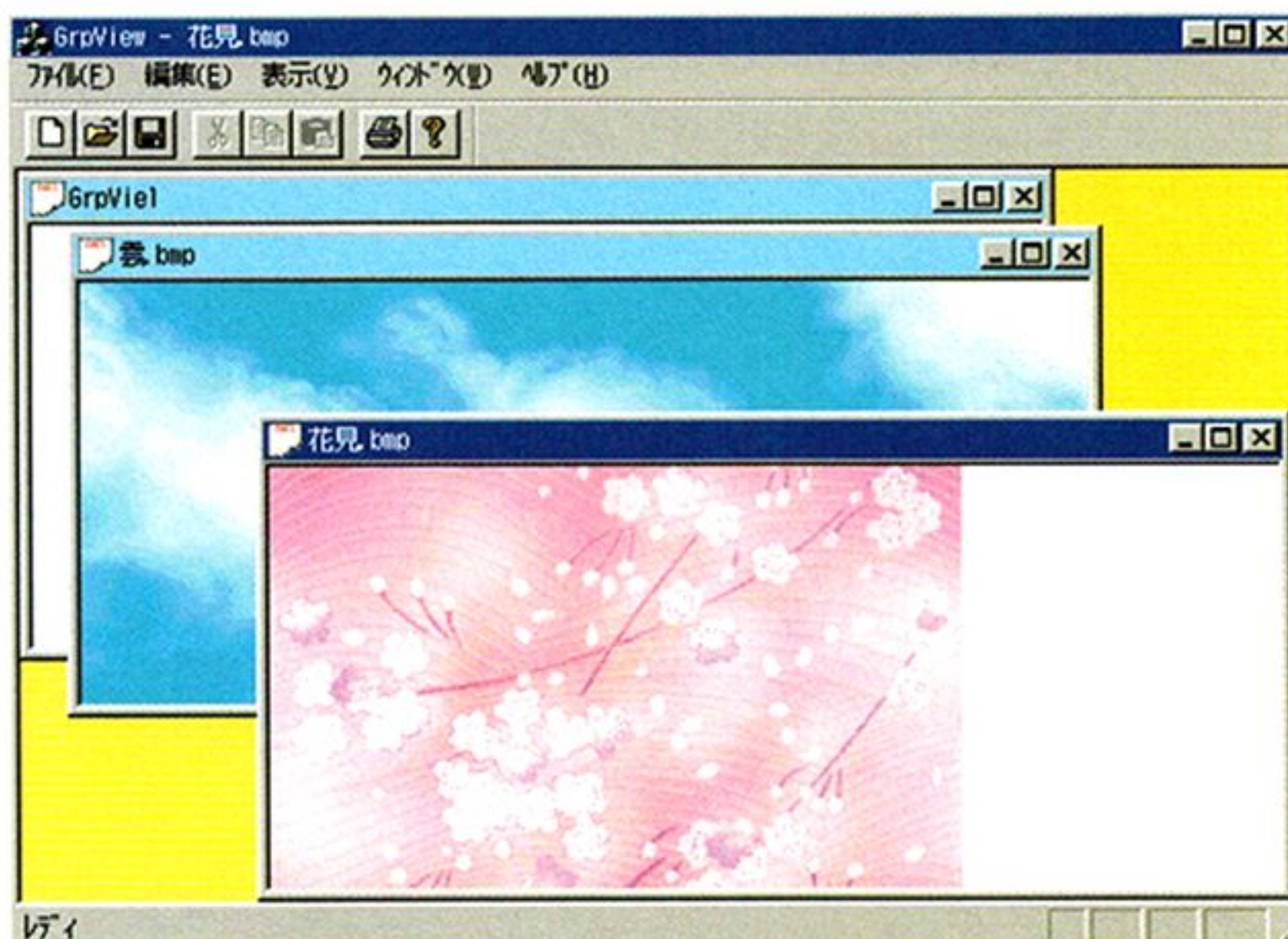
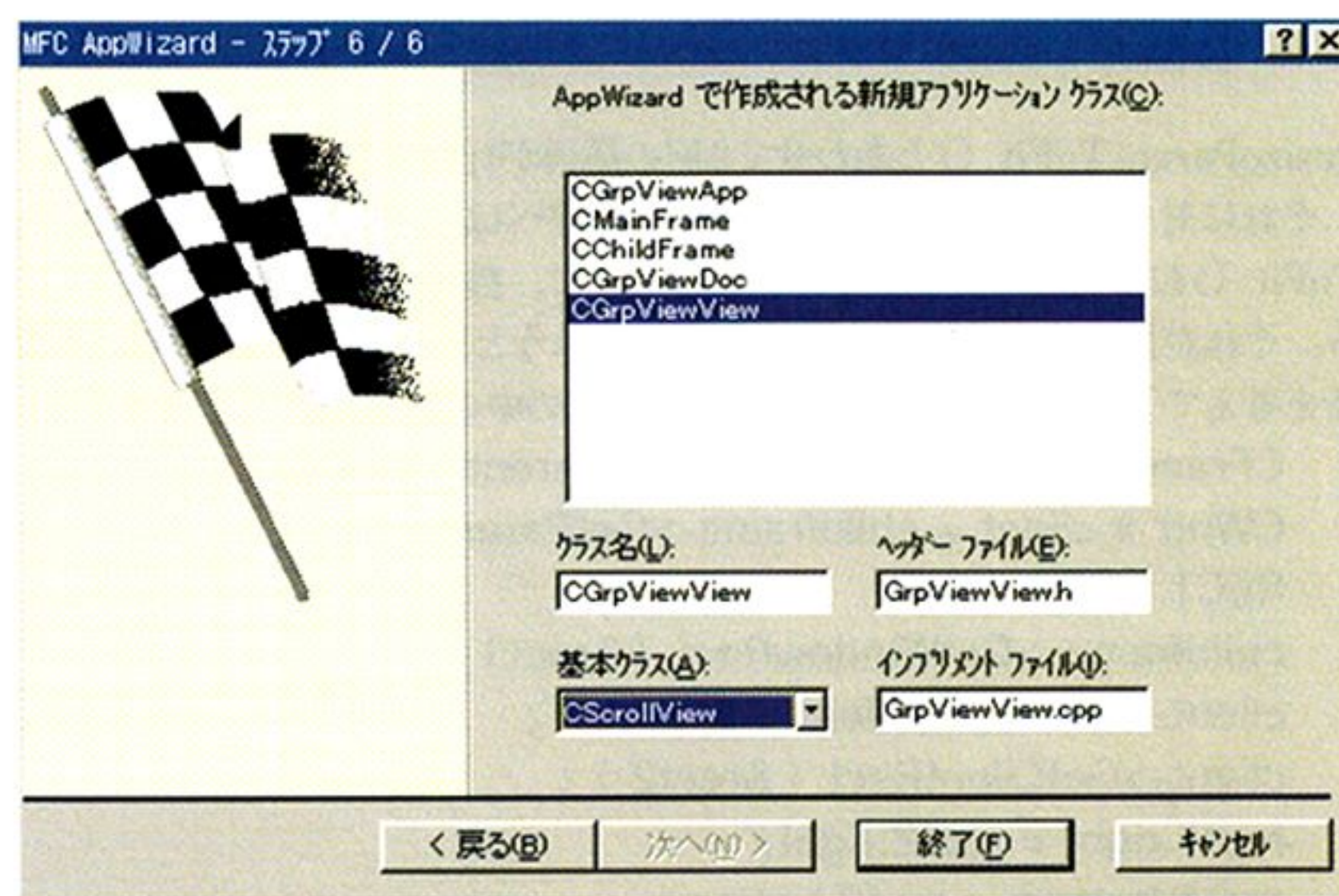
スクロールバーについてだが，これは実はCViewの派生クラスであるCScrollViewクラスをCGrpViewViewクラスの基本クラスにすればよかったのだ。なんだか複雑だが，要はMFCにはスクロールバーを持ったビューのクラスが用意されているということだ。ちょっと話は戻るが，プロジェクトの作成時，AppWizardのステップ6で，CGrpViewViewを選択し，基本クラスでCScrollViewを選択すればよかったのだ（図10）。最初からプロジェクトを作り直してもいいのだが，たいした変更もないので，いまあるものに手を加えよう。とりあえず，GrpViewView.cppとGrpViewView.h内の〝CView″という文字列を〝CScrollView″に置換する。そうしたら，あとはプロテクトメンバである次のメソッドをオーバーライトするだけだ。

```
virtual void OnInitialUpdate () ;
```

このメソッド内で，次のようにドキュメントのサイズを指定してやる。

```
CSize sizeTotal;
CGrpViewDoc * pDoc = GetDocument () ;
if ( pDoc->m_hDib ) {
  LPSTR lpDIB = (LPSTR) GlobalLock ( pDoc->m_hDib ) ;
  DWORD width = DIBWidth ( lpDIB ) ;
  DWORD height = DIBHeight ( lpDIB ) ;
  GlobalUnlock ( pDoc->m_hDib ) ;
  sizeTotal.cx = width;
  sizeTotal.cy = height;
  SetScrollSizes (MM_TEXT, sizeTotal) ;
}
```

SetScrollSizes（）の第1引数は単位を示すものであり，MM_TEXTはピクセルを論理単位とすることを示している。

ついでなので，画像が開かれたときに，子フレームのサイズを画像にあうように調整してやろう。CScrollViewクラスには，ResizeParentToFit（）というメソッドが用意されている。これはまさしくいまやろうとしていることのためのメソッドなのだが，少々厄介な点がある。このメソッド，BOOL型の引数を取るのだが，これがTRUEのときはウィンドウの縮小だけを行い，ドキュメントがフレームよりも大きい場合はウィンドウの拡大はやってくれない。というのは，むやみにドキュメントにあわせてフレームを拡大すれば，親フレームの中に入りきらなくなる場合があるからだ。これに対処する方法は3つある。

① こんなメソッドに頼らずに，自分でフレーム座標を算出する
② まずResizeParentToFit（FALSE）でフレームをドキュメントサイズに拡大しておいてから，もし親フレームをはみ出していたらマニュアルで調整する
③ フレームサイズを親フレームいっぱいの大きさにリサイズしておいてから，ResizeParentToFit（TRUE）をコールする

どうせなにかしらマニュアル調整が必要ならば，潔く①の方法というのが美しいが，タイトルバーやフレームの太さ，メニューの幅などを考慮に入れなければならず，意外に面倒臭い。そこで，ここでは②か③の方法を用いることにする。ウィンドウの大きさを変えるには，次のメソッドを使う。

```
void CWnd::MoveWindow ( LPCRECT lpRect,BOOL
                          bRepaint=TRUE ) ;
```

要はこのメソッドとResizeParentToFit（）のどちらを先に使うかということなのだが，MoveWindow（）の第2引数を見てもらいたい。このBOOL値は，ウィンドウを動かしたあとにリペイントするかどうかのフラグだ。もしこの関数をあとにした場合は，リペイントが必要となり，

図11　サンプルGrpView2

図12　GrpView-3

図13　GrpView-4

ResizeParentToFit ()とあわせて2回の描画が行われることになる。

それに対し，先にMoveWindow ()を呼べば，あとでResizeParentToFit ()によってリペイントが行われるので，描画は一度でいいことになる。それが目に見えたり，体感できるかというと疑問だが，精神的な合理性を考えて③の方法を採用し，先ほどのif文の中の最後に以下を追加する。

```
CFrameWnd * childframe = GetParentFrame () ;
CWnd * client = childframe->GetParent () ;
RECT rect1, rect2;
childframe->GetWindowRect ( &rect1 ) ;
client->ScreenToClient ( &rect1 ) ;
client->GetClientRect ( &rect2 ) ;
rect1.right = rect2.right;
rect1.bottom = rect2.bottom;
childframe->MoveWindow ( &rect1, FALSE ) ;
ResizeParentToFit ( TRUE ) ;
```

こういうことはCChildFrame側でやるべきなのだろうが，自分でやるときにはそうしてもらいたい。childframeというのは子フレーム，clientというのは親フレームの中の子フレームが自由に動ける作業領域だ。まず子フレームのウィンドウ領域をGetWindowRect ()で取得するが，これはスクリーン座標であるのでScreenToClient ()でクライアント座標に変換する。作業領域はGetClientRect ()でそのクライアント領域を取得するのだが，こちらはクライアント座標であるので，そのままでよい。座標が取得できたら，それぞれの領域の右と下端をあわせて，MoveWindow ()でサイズを変える。このMoveWindow ()もクライアント座標を引数に取るので，そのまま渡してしまえばよい。最後にResizeParentToFit ()で，ドキュメントが小さい場合に限りフレームを縮小する。これでOKだ。

さて，ビルドしてみよう(図11　サンプルGrpView2)。今度は起動時には空の子フレームは開かなくなった。少し大き目の画像を開いてみると，ちゃんと親フレームからはみ出ない程度にフレームのサイズが調整されていることがわかるだろう(図12　GrpView-3)。

パレットのマップ

まっとうに表示できるようになったかに見えるが，実はもうひとつ大切な作業がある。最近はハイカラー以上の環境が当たり前になっているし，画像を扱おうなんてアプリケーションがそこまで面倒を見る必要はないというのが筆者の持論ではあるが，触れないわけにはいくまい。256色環境への対応である。

いまのままでは，256色環境で画像を表示した場合，図のような悲しい結果となってしまう(図13　GrpView-4)。早い話が，表示できる色数が不足しているわけだ。しかし，Windowsの256色環境では，いわゆるパレット方式を採用している。フルカラー中の任意の256色を選択して表示できるということだ。したがって，表示したい画像に最適なパレットを選択すれば，それなりに256色環境でも綺麗に画像を表示できないこともない。ただ，アプリケーションがおのおの勝手にパレットを変更すると混沌としてしまうので，ある程度の規則，というかセオリーがある。

まず，パレットの選択といっても，ユーザーがシステムのパレットを直接編集できるわけではない。こんなパレットがほしーなーという見本を渡し，あとはデバイスコンテキストにお願いするのだ。デバイスコンテキストはいろいろなアプリケーションからのこのお願いを聞かなければならないので，必ずしも希望どおりにはならないのだが，「私はアクティブウィンドウだから最優先してね」というお願いもできる。また，そのお願いをするべきタイミングを計るメッセージも用意されている。WM_PALETTECHANGEDとWM_QUERYNEWPALETTEだ。前者はシステムパレットが誰か(自分も含む)によって変更されたときに，後者は「いまから君にフォーカスを渡すけど，その前にパレットを変更したいかい？」というチャンスを与えられたときにシステムから渡されるメッセージだ。この2つはCMainFrameクラスでハンドルできる。さらにMDIである場合には，子フレームがアクティブになったときにもパレット変更の要求が必要となる。これは，ビュークラスのOnActivateView ()メソッドをオーバーライドすれば，そのタイミングは取得できる。ではインプリメントしてみよう。

まずは画像側のパレットの取得だ。これは，例のDIBAPIの中のCreateDIBPalette ()関数で取得できる。何度か使うことになるので，ド

キュメントクラスにCPaletteクラスの変数を用意し，画像を開くOnOpenDocument（）メソッド内で取得しておくといいだろう。こちらもm_hDib同様publicにしておく。ヘッダ内で，

```
CPalette ＊m_pPal;
```

として宣言したなら，

```
m_pPal = new CPalette;
CreateDIBPalette（m_hDib, m_pPal）;
```

とすればいい。コンストラクタで初期化，デストラクタで廃棄も忘れないように。

そうしたら，今度はWM_PALETTECHANGEDとWM_QUERYNEWPALETTEをClassWizardを使ってハンドルする。この辺の説明はもういいだろう。さて，実際にパレットの変更要求を出すのはビュークラスが適切だと思われるのだが，問題となるのはCMainFrameクラスからどうやってビュークラスを見つけるかだ。

ヘルプを探してみると，MDIGetActive（）とGetActiveView（）を使えば，とりあえずアクティブなビューは取得できるが，アクティブでない子ウィンドウのビューが取得できない。GetNextWindow（）などを使えば取得できないこともないのだが，少々面倒なので，ユーザーメッセージを送ることにしよう。いままではメッセージといえば天から送られてくるものだったのだが，今度は自分でメッセージを作って，ビューに向けて送ってみようということだ。基本的にメッセージは2つのパラメータを渡すことができ，SendMessage（）関数などを使って送ることができる。

```
LRESULT SendMessage（HWND hWnd, UINT Msg,
                    WPARAM wParam, LPARAM lParam）;
```

hWndはメッセージの送り先のウィンドウハンドルを，Msgはメッセージを示す。wParamとlParamはなにか渡すものがあれば自由に使っていいよというパラメータで，メッセージによって決定される。たとえば，WM_LBUTTONDOWNというメッセージは，wParamにはキーのフラグが，lParamの上位ワード，下位ワードにはそれぞれX，Y座標が入る。これをMFCを使ってハンドルすると，

```
void OnLButtonDown（UINT nFlags, CPoint point）;
```

となってしまうが，これはMFCが人知れずパラメータの形式のわかっているものは勝手に組み替えてしまうからだ。

さて，ユーザーメッセージであるが，勝手にメッセージを作ってしまうと，ほかのシステムのメッセージとぶつかってしまう危険性がある。しかし，Windowsでは，そういった場合のためにユーザーが使ってよいメッセージ領域を用意してある。WM_USERからがそうだ。複数定義したい場合にはWM_USER+1,WM_USER+2……として使っていくことができる。ここではとりあえず，GrpView.hで以下のように定義しておく。

```
#define WM_REALIZEPALETTE WM_USER
```

では，WM_PALETTECHANGEDとWM_QUERYNEWPALETTEのメッセージ関数OnPaletteChanged（）とOnQueryNewPalette（）に戻ろう。先ほどのSendMessage（）は特定のウィンドウにメッセージを送る関数だが，手っ取り早くすべてのビューにメッセージを送るには，SendMessageToDescendants（）という，すべての子ウィンドウに対してメッセージを送る関数を使う。余計なウィンドウにもメッセージが送られる可能性もあるが，WM_REALIZEPALETTEをハンドルしていないウィンドウでは無視されるだけだ。ただし，OnPaletteChanged（）ではすべてのビューに対して「パレット要求していいよ」メッセージを出すが，OnQueryNewPalette（）ではアクティブビューに対してしかメッセージを出さない。理由は，OnQueryNewPalette（）でパレットの変更が行われれば，そのあとWM_PALETTECHANGEDメッセージも送られてくるからだ。というわけで，この2つのメッセージ関数は次のようになる。

```
void CMainFrame::OnPaletteChanged（CWnd ＊
                                  pFocusWnd）
{
  CMDIFrameWnd::OnPaletteChanged（pFocusWnd）;
  CMDIChildWnd ＊pChild = MDIGetActive（）;
  if（pChild==NULL） return;
  CView ＊pView = pChild->GetActiveView（）;
  SendMessageToDescendants（WM_REALIZEPALETTE,
(WPARAM）pView->m_hWnd）;
}

BOOL CMainFrame::OnQueryNewPalette（）
{
  CMDIChildWnd ＊pChild = MDIGetActive（）;
  if（pChild==NULL） return FALSE;
  CView ＊pView = pChild->GetActiveView（）;
  pView->SendMessage（WM_REALIZEPALETTE,
                     (WPARAM）pView->m_hWnd）;
  return TRUE;
}
```

ここで，WM_REALIZEPALETTEメッセージのwParamとして，アクティブビューのウィンドウハンドルを渡すことにする。というのは，ビューがパレットの要求をするときに，自分がアクティブかどうかを調べるためだ。もしこのパラメータとビュー自身のウィンドウハンドルが同じならば，自分はアクティブであると判断できる。

今度はビュークラスにメッセージハンドラを作る。どうやらユーザーメッセージの場合はClassWizardは使えないので（ひょっとしたら可能なのかもしれないが，筆者は方法を知らない），手動で行う必要があるようだ。まずはCViewView.hの中の，

```
//{{AFX_MSG（CGrpViewView）
```

と，

```
//}}AFX_MSG
```

で挟まれた部分に，次の1行を差し込む。

```
afx_msg LRESULT OnRealizePalette（WPARAM wParam,
                                 LPARAM lParam）;
```

ここに書き込んでやると，このメソッドはメッセージ関数であるということをClassWizardやMFCが認識してくれる。さらに，CViewView.cppのBEGIN_MESSAGE_MAP（CGrpViewView, CScrollView）の中の，

```
//{{AFX_MSG_MAP（CGrpViewView）
```

と，

```
//}}AFX_MSG_MAP
```

で挟まれた部分に，

```
ON_MESSAGE（WM_REALIZEPALETTE, OnRealizePalette）
```

を差し込んでやる。ここも，もしWM_REALIZEPALETTEというメッセージが来たら，OnRealizePalette（）を呼びなさい，ということをMFCに指示している部分だ。

さて，肝心のOnRealizePalette（）関数だが，パレットの要求に使うデバイスコンテキストとして，ビューのClientDCを使ってみた。その名のとおり，クライアント部分を描画するためのデバイスコンテキストだ。初期化はコンストラクタで行うことができ，引数としてウィンドウクラスを渡す。

```
CClientDC DC（this）;
```

やっとthisが出てきたね。覚えているか？　これは自分自身を指し示すポインタ，すなわちここではビュークラスのインスタンスを示していることになる。つまり，ビューのクライアントのデバイスコンテキストができるわけだ。このデバイスコンテキストに対してドキュメントのパレットを選択する。

```
＊pOldPalette = DC.SelectPalette（pDoc->m_pPal,
                  ((HWND）wParam）!=m_hWnd）;
```

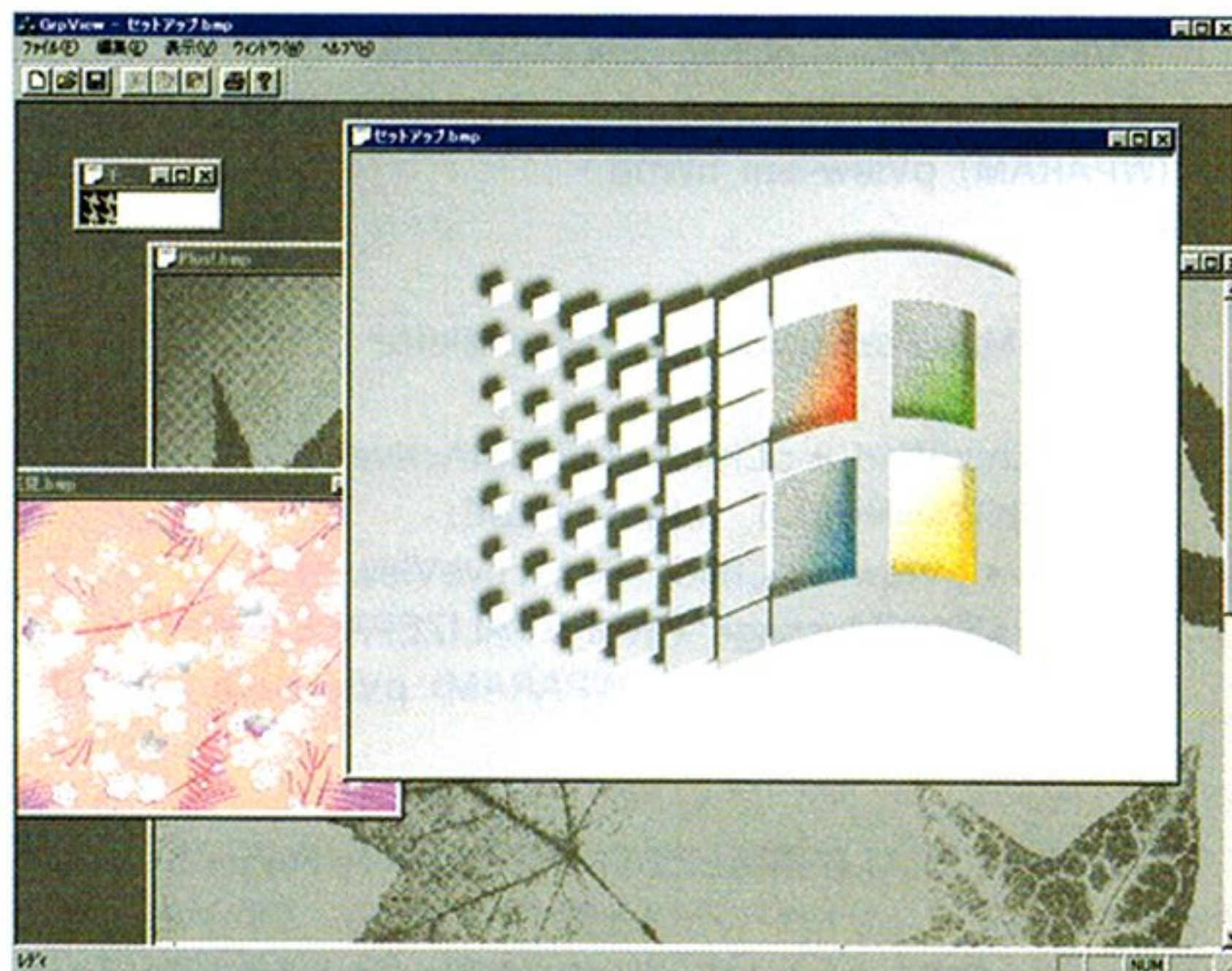
SelectPalette（）の第2引数には，バックグラウンドパレットにするかどうかを指定する。パレットがフォアグラウンド，つまり「アクティブウィンドウだから優先してね」のときにはFALSEを指定しなければならないので，wParamで渡されたアクティブなウィンドウハンドルと自分のウィンドウハンドルを比較し，もし違っていたら，つまりアクティブでなかったらTRUEを，そうでなければFALSEを設定するようにしている。

```
DC.RealizePalette（）;
```

が「適当にパレットをシステムにマップしてね」要求だ。そのあと，

```
DC.SelectPalette（pOldPal, TRUE）;
```

図14　サンプルGrpView-3

図15　GrpView-5

で元のパレットに選択を戻している。前回も少し述べたが，デバイスコンテキストはいつ誰が使うかわからないので，使い終わったあとは元に戻しておかなければならない。

さて，これでパレットはマップされたのだが，最後にビューを更新し，そのパレットに従って再描画しなければならない。これにはInvalidate ()メソッドを使おう。直接ビューを書き換えるのではなく，クライアント領域を無効にするメソッドだ。無効になったクライアントを再び有効にするために，あとで暇なときにWM_PAINTメッセージを発行し，クライアントを更新させる。暇なときというのは，メッセージがアイドル状態になったときだが，そうはいっても人間から見れば瞬時だ。

もう1カ所パレットを変更するチャンスが残っていた。子フレームがアクティブになったときだ。OnActivateView ()をオーバーライドしたら，OnQueryNewPalette ()同様パレットを変更するのはアクティブになったフレームだけでいいだろう。今度はわざわざ回りくどいメッセージを使うまでもない。直接，

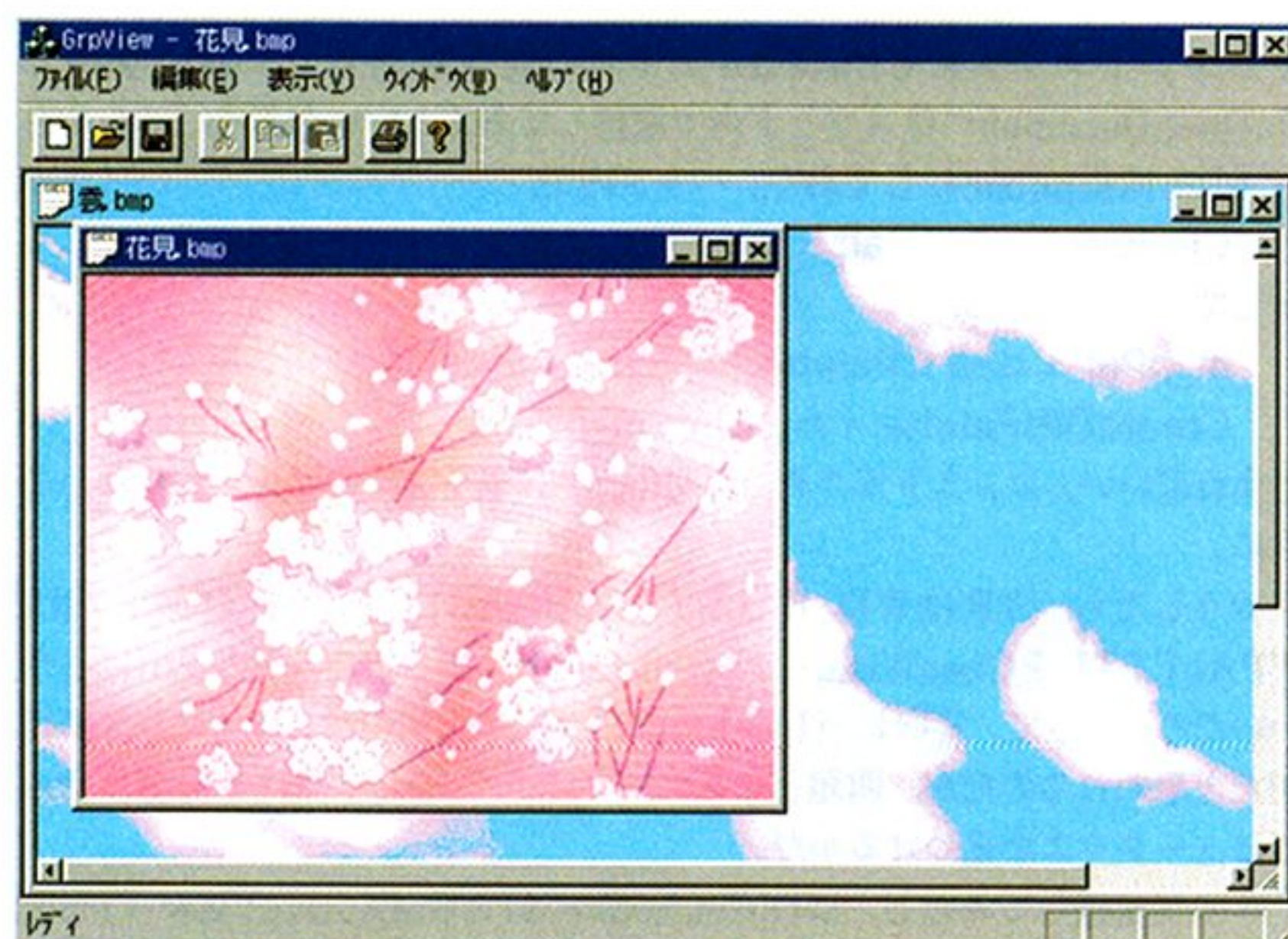
```
OnRealizePalette（（WPARAM）m_hWnd, 0）;
```

としてパレット更新の関数を読んでやればいいだろう。

さて最後は描画時だ。描画を行うときにもパレットをデバイスコンテキストに選択してやらなければならないのだが，それはさっきはとりあえずNULLにしたPaintDIB ()の最後の引数にパレットを指定すればよい。PaintDIB ()の内部でデバイスコンテキストにSelectPaletteされ，最適に描画を行ってくれるはずだ(図14　サンプルGrpView-3)。2つの画像を開くと，アクティブな画像は綺麗に，非アクティブな画像も極端に色化けせずに表示できているのがわかるだろう(図15　GrpView-5)。それでも「雲」は少々辛いが，これは「花見」がパレットを目一杯使ってしまっているためで，これ以上はいたしかたないといったところだ。

えらそうなことをいった割には，実は筆者もまじめにパレット周りをいじったことがなく，この辺りはDIBLOOKサンプルを参考にした。ヘルプはなにをいいたいのか意味不明だし。しかも「256色でも綺麗に表示できる」といいながら，それは結局256色以下のBMPを読み込んだときの話だ。それよりも色数の多い画像からはパレットが作れないから。もちろん自分で最適な色を選び出してパレットを作れば対処できるはずなので，もしどうしても256色環境でフルカラー画像を綺麗に，というのならば自分でトライしてもらいたい。「なんだかんだいって，DIBLOOKサンプルとたいして変わらんじゃないか」という君，まあそうあせらないで，もうちょっといじっていくから。

落書きしてみよう

まあ一応ビュアとしての最低限の機能はできた。拡大とかやりたければ自分でやってもらうとして(PaintDIB ()に渡すRECT構造体次第ってのはわかるよね)，今度はちょっとGrpViewという名に反して書き込みなんかもやってみよう。まあ，前回も似たようなことをしたけど，今度はドロー系じゃなくてペイント系ということで。前回はデバイスコンテキストの描画関数によって，ウィンドウに直接描画を行った。

今回もデバイスコンテキストの描画関数を使うのだが，前回とはちょっと違う。というのは，今回はメモリ上のビットマップデータに対して描画を行う必要があるからだ。これにはちょっと段階が必要となる。まずデバイスコンテキストはデバイスコンテキストでも，メモリデバイスコンテキストという，いってみれば仮想的なデバイスコンテキストをメモリ上に作成する。次にDIBデータからビットマップを作成してハンドルを取得し，それとメモリデバイスコンテキストを関連付ける。そこまでできれば，あとはメモリデバイスコンテキストに描画を行えば，それに関連付けられているビットマップに書き込まれるというわけだ。とりあえずその辺りはOnOpenDocument ()で処理してしまおう。まずはメモリデバイスコンテキストをドキュメントクラスのメンバとして登録しよう。

```
CDC ＊m_pDC;
```

そうしたら，次のように作成する。

```
m_pDC = new CDC;
m_pDC->CreateCompatibleDC（NULL）;
```

これだけで，もうメモリデバイスコンテキストはできてしまった。次に，DIBからビットマップを起こす。ちなみにDIBと(ここでいう)ビットマップの決定的な違いは，DIBがデバイス非依存ということだ(DIBとはDevice Independent Bitmapの略)。デバイスというのは，この場合ビデオドライバなりビデオチップということになる(あるいはプリンタに出力するならプリンタドライバ)。DIBにはヘッダがついていて，そのヘッダを見ればビットカウントがわかるし，ビット配列も決まっている。しかし，ビットマップはそのときのWindowsの環境，ビデオドライバや表示色によって内部構造は異なっている可能性がある。したがって，ユーザーはビットマップの内部にはおいそれと直接手が出せないように，ハンドルで管理される。話がそれたが，ビットマップを作ろう。これには次のWin32APIを使う。

```
HBITMAP CreateDIBitmap（HDC hdc, CONST
BITMAPINFOHEADER ＊lpbmih, DWORD fdwInit, CONST
VOID ＊lpbInit, CONST BITMAPINFO ＊lpbmi, UINT
fuUsage）;
```

CDCクラスにあってもよさそうなものだが，MFCではサポートしていない。したがって，戻り値もCBitmapクラスではなく，ハンドルそのものを扱うことになる。まず第1引数だが，これは依存するデバイスコンテキストを指定する。デバイス非依存のものを依存のものに変換するわけだから，その基準となるデバイスの指定ということだ。第2引数のBITMAPINFOHEADERは，DIBの先頭にくっついているものなので，ReadDIBFile ()で取得したメモリハンドルをGlobalLock ()したものを渡せばよい。

次は初期化フラグだ。与えられたビット列でビットマップが初期化されるかどうかを指定する。初期化するならCBM_INITとなる。その次はビット列のポインタだ。これはDIBAPIにあったFindDIBBits（）で取得できる。BITMAPINFOには，BITMAPINFOHEADERと同じポインタを指定する。というのは，BITMAPINFOはBITMAPINFOHEADERの後ろにパレット情報をくっつけた構造体なのだ。同じポインタを2カ所で指定させるというのも不可解だが，おそらく最初のBITMAPINFOHEADERはビットマップの作成時に，BITMAPINFOは初期化時に使われるのだろう。最後のフラグは，カラーテーブルをどこから取得するかを示し，DIBデータからならDIB_RGB_COLORSでよい。

ただし，このCreateDIBitmap（）の前にすることがある。例によって，パレットの処理だ。どうやらこの関数を呼ぶときにhdcに選択されていたパレットでビットマップが作成されるようなので，まずはそちらを処理しなければならない。ちなみにデバイスコンテキストは，親フレームのClientDCで指定してみる。実はぶっちゃけた話，デバイスコンテキストというものはシステムディスプレイ，つまり画面表示されているものはすべて同じものだと思ってよい。取得するときに，原点やクリッピングエリアがウィンドウごとに設定されるだけなのだ。したがって，ここで使うのは，子フレームのデバイスコンテキストでも，デスクトップウィンドウのデバイスコンテキストでも構わない。

```
LPSTR lpDIBHdr = (LPSTR) GlobalLock ( hDib ) ;
LPSTR lpDIBBits = FindDIBBits ( lpDIBHdr ) ;
CClientDC DC ( AfxGetApp () ->m_pMainWnd ) ;
CPalette * pOldPal = DC.SelectPalette ( m_pPal, TRUE ) ;
DC.RealizePalette () ;
m_hBitmap = CreateDIBitmap ( DC.m_hDC,
        (BITMAPINFOHEADER *) lpDIBHdr, CBM_INIT,
lpDIBBits, (BITMAPINFO *) lpDIBHdr, DIB_RGB_COLORS
                                                  ) ;
DC.SelectPalette ( pOldPal, TRUE ) ;
m_nWidth = DIBWidth ( lpDIBHdr ) ;
m_nHeight = DIBHeight ( lpDIBHdr ) ;
GlobalUnlock ( hDib ) ;
GlobalFree ( hDib ) ;
```

ただし，パレットはSelectPalette（）しただけではダメで，RealizePalette（）までしなければならないらしい。ついでなので，幅と高さを保存するm_nWidthとm_nHeightもメンバに加え，取得しておこう。ビットマップが完成したら，もうDIBは用済みなので解放して構わない。最後に，

```
m_hOldBitmap = (HBITMAP) SelectObject ( m_pDC
                          ->m_hDC, m_hBitmap ) ;
```

としてビットマップをデバイスコンテキストに選択する。ビットマップがハンドルなので，ここではCDCのメソッドでなく，Win32APIを使っている。

ここまでできればあとは簡単だ。ビュークラスのOnDraw（）で，今度はデバイスコンテキスト同士の転送をBitBlt（）というメソッドで行えばよい。

```
BOOL CDC::BitBlt ( int x, int y, int nWidth, int nHeight,
     CDC * pSrcDC, int xSrc, int ySrc, DWORD dwRop ) ;
```

xとyは転送先の原点，nWidth，nHeightは大きさ，pSrcDCは転送元デバイスコンテキスト，xSrcとySrcは転送元の原点を示す。最後はラスタオペレーションコードといい，転送のビット操作を指定できる。単純にコピーするだけなら，SRCCOPYとなる。ただし，またまたここでも転送前に，転送先のデバイスコンテキストにパレットを選択させ，RealizePalette（）させなくてはならないようだ。三度手間になるが，これでちゃんと表示されているようなので，よしとしよう。

```
CPalette * pOldPal = pDC->SelectPalette ( pDoc
                              ->m_pPal, TRUE ) ;
pDC->RealizePalette () ;
pDC->BitBlt ( 0, 0, pDoc->m_nWidth, pDoc->m_nHeight,
                   pDoc->m_pDC, 0, 0, SRCCOPY ) ;
pDC->SelectPalette ( pOldPal, TRUE ) ;
```

ここでひと息つきたいところだが，もうちょっと我慢。我々には落書きをするという崇高な目的があったのだ。とはいっても，前回を覚えていれば難しいことはあるまい。OnLButtonDown（），OnLButtonUp（），OnMouseMove（）をハンドルし，適当に先ほど取得したメモリデバイスコンテキストに書き込んでやればいいのだ。前回と同じだが，またMoveTo（）とLineTo（）でラインを引いてみた。ただし，今回はアンカーポイントを保存していない。なぜなら，このメモリデバイスコンテキストは自分以外には使われることがないからだ。したがって，前回の座標を確実に次回まで保持していてくれる。前回よりも圧倒的に簡単だろう。

ただし，注意が必要なのは，書き込んでいるデバイスコンテキストはあくまでもメモリデバイスコンテキストで，そのままでは画面に反映されないということだ。そのため，LineTo（）で書き込んだあとは，InvalidateRect（）でクライアントを無効化し，あとでWM_PAINTにより画面が更新されるようにしている。

```
void CWnd::InvalidateRect ( LPCRECT lpRect, BOOL
                            bErase = TRUE ) ;
```

さっき使ったInvalidate（）と違うのは，無効化する領域を指定できるかどうかだ。面倒なのでとりあえずNULLとし，全領域書き換えにしてしまっているのでInvalidate（）でも違いはないのだが，描いた領域だけを書き換えるように変更すれば，多少の効率化にはなるだろう。

また，第2引数のフラグは，背景を消去するかを示すフラグだ。これをTRUEにしておくと，更新時にまずウィンドウの背景色（おそらくニュートラルグレー）で塗りつぶしてから，OnDraw（）が呼ばれる。今回はその必要はない，というよりも，いちいち背景で塗りつぶされるとちらつくので，FALSEにしておこう。

勢いに乗って，セーブまでやってしまおう。DIBAPIのSaveDIB（）はDIBをファイルに保存するための関数なので，まずビットマップをDIBに戻さなければならない。それには，また例によってサンプルを流用させてもらうことにしよう。

WriteAVIサンプルにちょうどいい関数が用意されていた。WriteAVI.Cのいちばん最後，MakeDib（）だ。この関数をMYFILE.CPPの最後にでも，プロトタイプ宣言をDIBAPI.Hにそれぞれ追加しよう。ただ，このMakeDib（）は内部でデバイスコンテキストを取得してDIBにしているため，またまたパレット情報が問題になってくる。そこで，外部からデバイスコンテキストを渡すように改良しよう。

```
HANDLE  MakeDib ( HDC hdc, HBITMAP hbitmap, UINT
                                      bits ) ;
```

として第1引数でhdcを渡すようにして，あとは内部の宣言と初期化，廃棄の部分をコメントアウトすればいい。第3引数のbitsというのは，DIBのビットカウントを指定する。ここには元のDIBのビットカウントを指定すべきだろうが，残念ながらDIBAPIにはビットカウントを取得する関数は用意されていないので，自分で作ることにしよう。

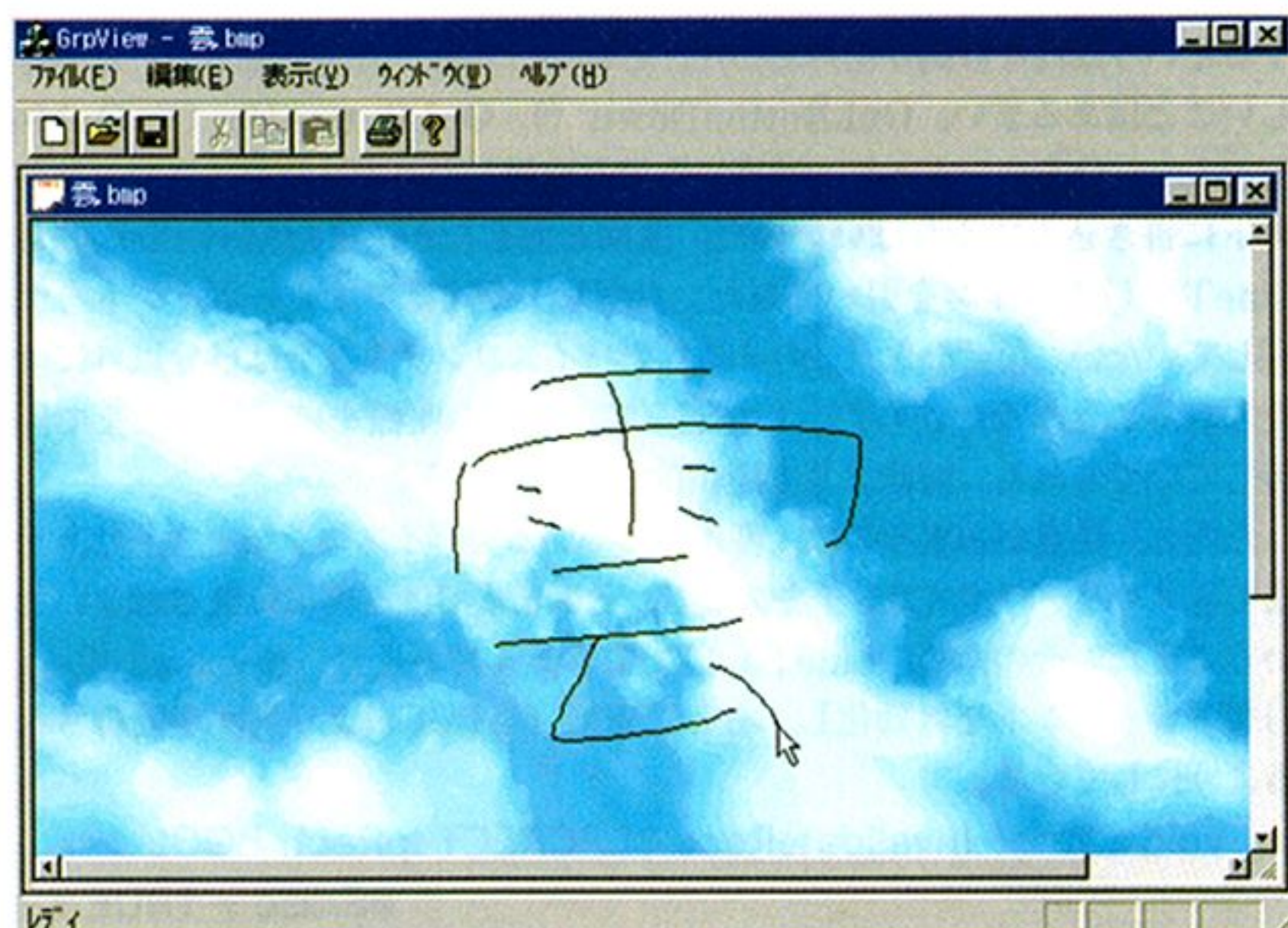
```
int GetBitCount (LPSTR lpbi)
{
  if (IS_WIN30_DIB (lpbi))
    return ((LPBITMAPINFOHEADER) lpbi) ->biBitCount;
  else
    return ((LPBITMAPCOREHEADER) lpbi) ->bcBitCount;
}
```

これをDIBAPI.CPPに追加し，DIBAPI.Hでプロトタイプ宣言する。要は，さっきも出てきたBITMAPINFOHEADER構造体からビットカウント情報を返しているだけだ。BITMAPCOREHEADERというのは古いタイプのDIBのヘッダで，IS_WIN30_DIB（）マクロはどちらのタイプかを判断してふるい分けている。これはドキュメントをオープンしたときに，ドキュメントクラスにメンバm_nBitCountを作って保存しておこう。

さて，セーブのほうだが，ドキュメントクラスのOnSaveDocument（）クラスをオーバーライドしたら，次のような感じでいいだろう。

```
BOOL result = FALSE;
CFile file ( lpszPathName, CFile::modeCreate|CFile::mode
                                         Write ) ;
CClientDC DC ( AfxGetApp () ->m_pMainWnd ) ;
```

図16　GrpView-6

図17　linepropdlg

```
CPalette ＊pOldPal = DC.SelectPalette ( m_pPal, TRUE ) ;
DC.RealizePalette () ;
HDIB hDib = (HDIB) MakeDib ( DC.m_hDC, m_hBitmap,
                              m_nBitCount ) ;
DC.SelectPalette ( pOldPal, TRUE ) ;
if ( hDib ) {
  result = SaveDIB ( hDib, file ) ;
  GlobalFree ( hDib ) ;
}
return result;
```

ほとんど説明はいらないだろう。例によって，デバイスコンテキストにSelectPalette ()，RealizePalette ()したら，さっきのMakeDib ()でビットマップからDIBを作成し，DIBAPIのSaveDIB ()関数でファイルに保存する。

これで「落書き・保存」までできるようになった（サンプルGrpView4，図16　GrpView-6)。ただし，問題がないわけじゃない。DIBをいったんビットマップにし，それをまたDIBに戻すわけだから，途中の変換で情報が落ちるのは避けられない。たとえば，256色環境でフルカラーBMPを読み込んだ場合，DIBからビットマップへの変換過程で8ビットカラーに落とされる。それをまたDIBに戻すわけだから，フォーマットが24ビットカラーでも，中身は実質8ビットカラーとなってしまう。これはハイカラー環境でも同じことがいえるので，それが気になるのであればフルカラー環境で使いなさいということだ。ま，サンプルだし，気にしないことにしよう。

線の色と太さ

前回は線の色を選択できるようにしたが，メニューを選択するとモーダルダイアログボックスが開き，そこから決められた色をラジオボタンで選択するという，なんとも中途半端な仕様止まりだった。今回はもうちょっとスマートに，モードレスダイアログボックスを使い，色をRGB値で指定でき，ついでに線の太さも選択できるようにしてみよう。それではまず，ダイアログのデザインから。

［挿入］－［リソース］からDialogを選択し，新規作成したら，図のようにコントロールを配置しよう（図17　linepropdlg)。もともとあった[OK]と［キャンセル］ボタンは削除してしまって構わない。RのスライダーIDはIDC_RSLIDER，エディットボックスはIDC_REDIT，スピンボタンはIDC_RSPINとしておき，G，Bも同様だ。

それぞれのコントロールのテキストの配置などは自由にやって構わないが，重要なのはスピンボタンのプロパティだ。［自動関連付け］と［数値の自動表示］にチェックをし（図18　spinprop)，タブオーダーは必ずそれぞれのエディットボックスの直後に設定しよう（図19　taborder)。こうしておけば，スピンボタンが操作されたときのメッセージをユーザーが拾って処理しなくても，自動的にエディットボックスに値を入れていってくれる。

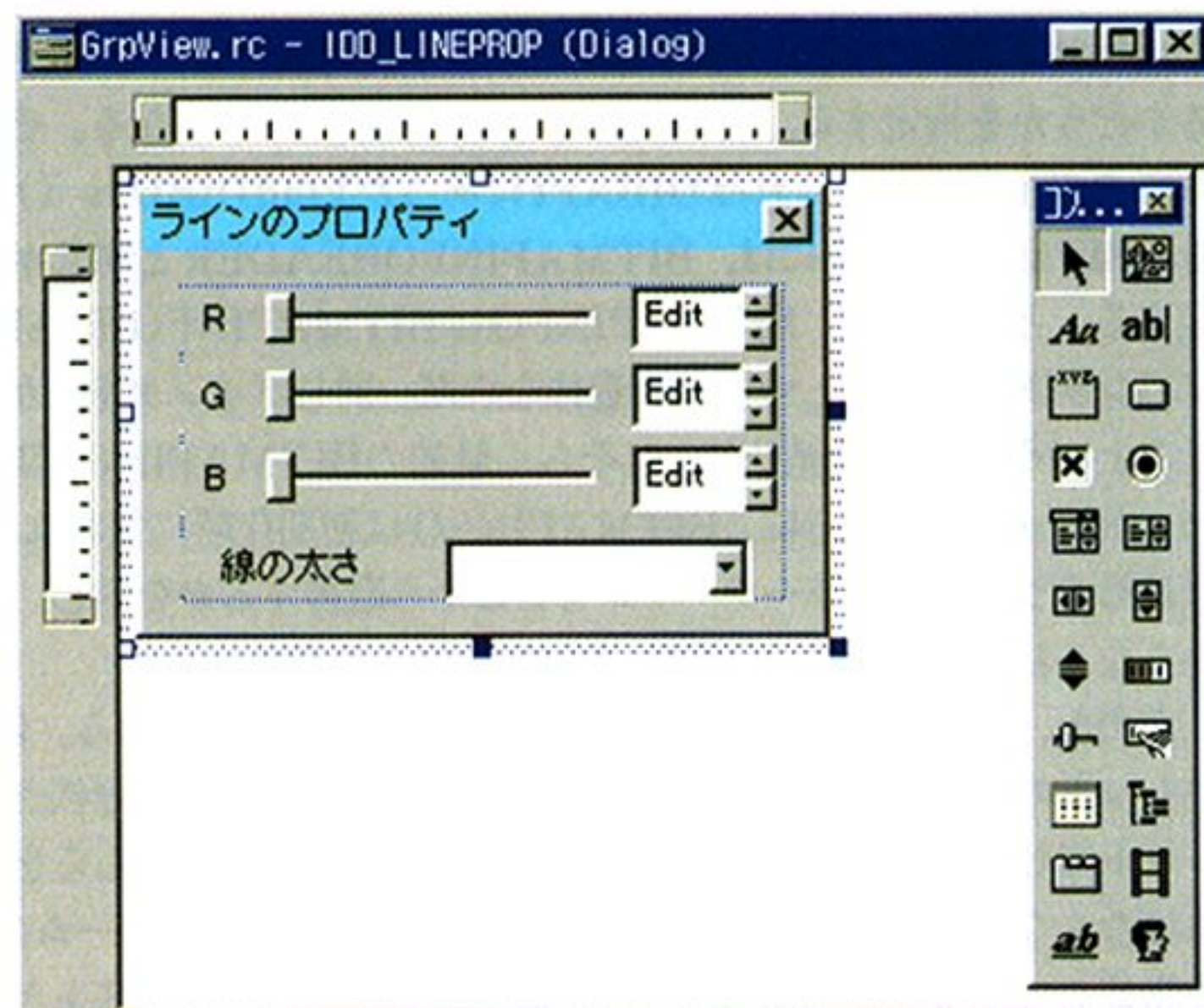
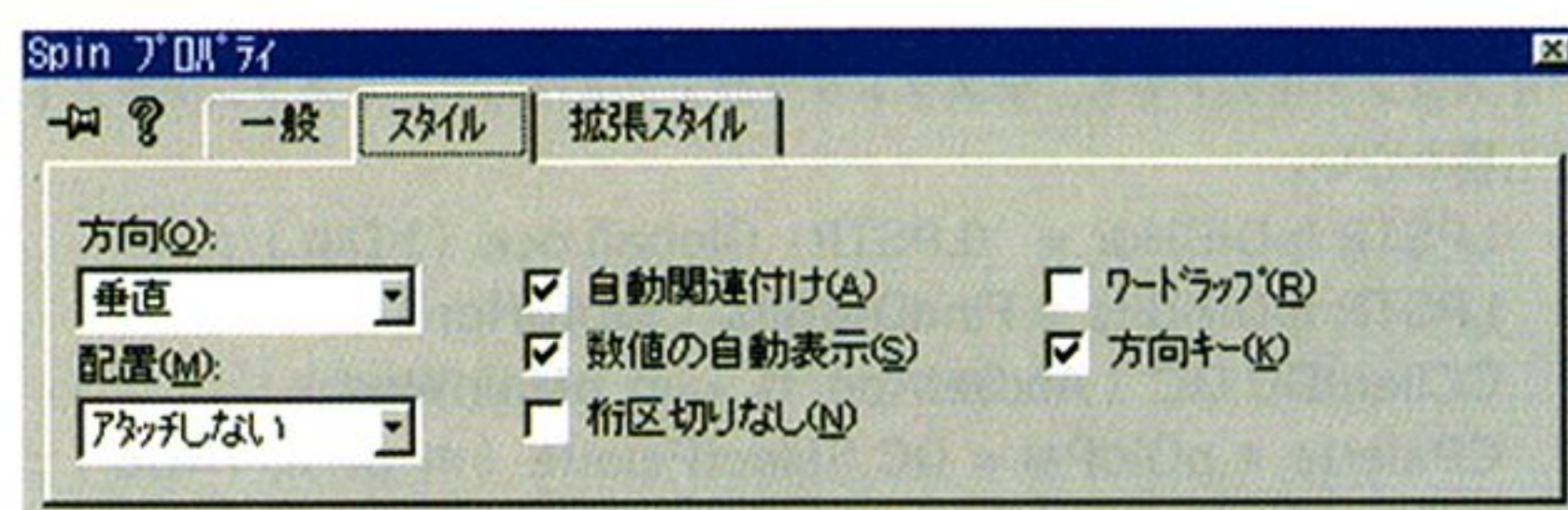
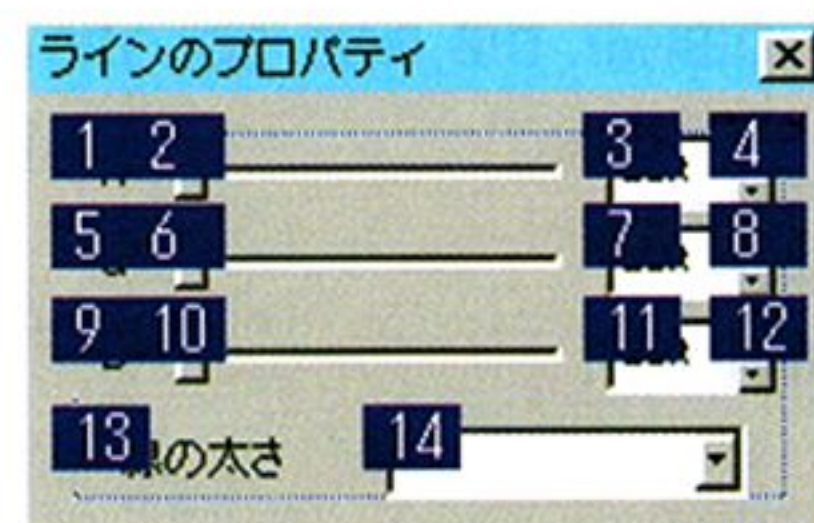



図18　spinprop

図19　taborder

また，コンボボックスには，初期データとして"1"，"2"，"3"，"4"くらいを入れておこう。ただし，これはあくまでもダミーで，リストは4つあるよ，くらいに思っておいてもらいたい。改行するには，Ctrlキーを押しながらリターンキーを押す（図20　combodata)。

スタイルではタイプを［ドロップダウンリスト］に，オーナー描画を［固定］にし，ソートのチェックを外しておく。ただし，先にタイプを指定してしまうと，ドロップダウンのサイズを指定できなくなってしまう。まずは［標準］のままで適当な大きさまでコントロールを広げる。このとき，ダイアログをはみ出して大きくすることはできないので，ダイアログも適当に大きくしておく（図21　combo-1)。そのあと，プロパティのタイプを変更（図22）し，最後にダイアログのサイズも調整するとよい。不思議なインタフェイスだ。［レイアウト］の［テスト］を実行してみると，どのくらいべろが伸びるのか確認できる。このとき，オーナー描画を［いいえ］に戻しておけば，入力したデータがリストに表示される（図23　combo-2)。

できたら今度はClassWizardからダイアログクラスを作ろう。前回と同じだ。［クラスの追加］－［新規...］を選択し，クラス名はCLinePropDlg，基本クラスはCDialog，IDをIDD_LINEPROP（さっきのダイアログテンプレートのID）とすればOKだ（図24　clinepropdlg)。これでクラスは追加された。そうしたらメンバ変数を追加していこう。エディットボックス，コンボボックスはint型の値で構わない。さらに，エディットボックスは値の範囲を0から255としておいていいだろう（図25　member)。

さて，このダイアログはどこで表示するようにしようか。やはりメニュー

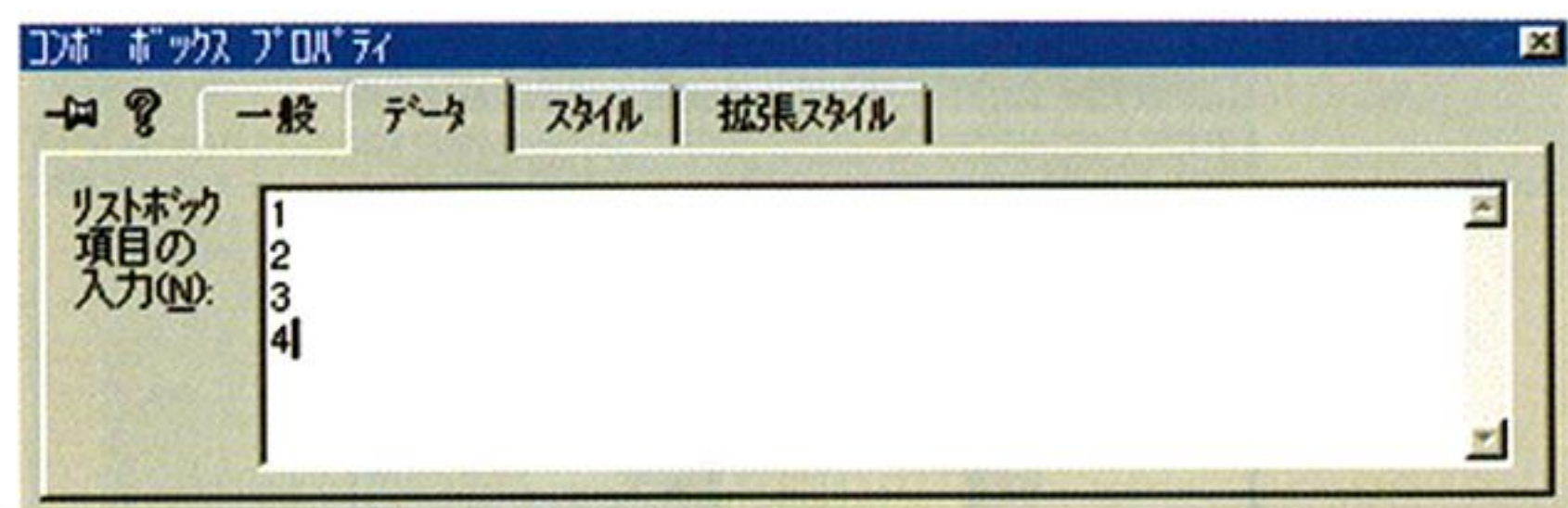



図20　combodata

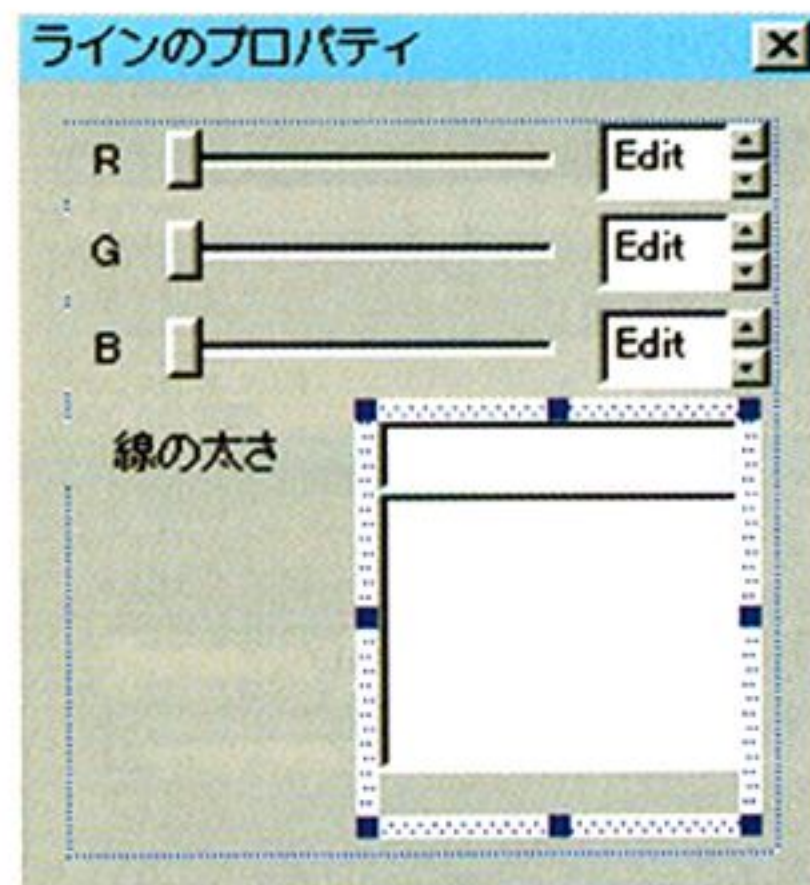



図21　combo-1

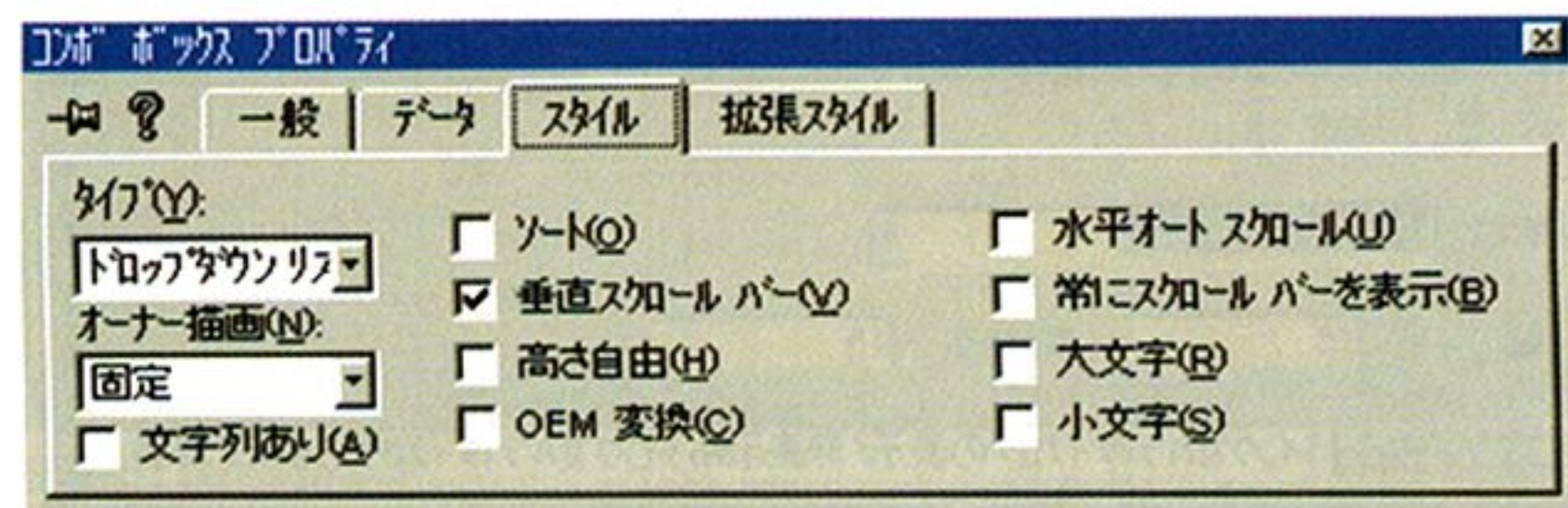



図22

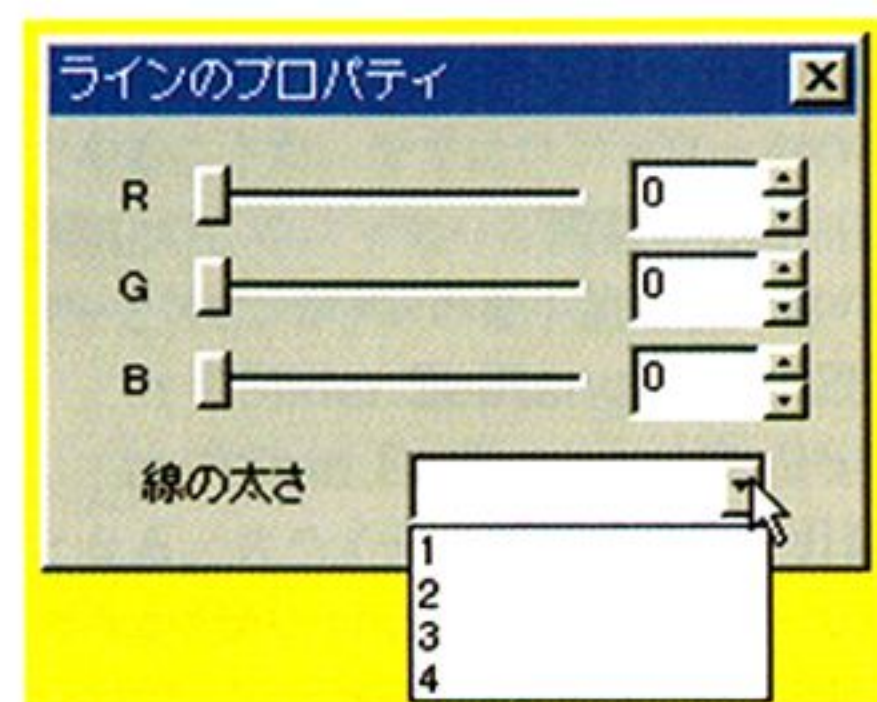



図23　combo-2

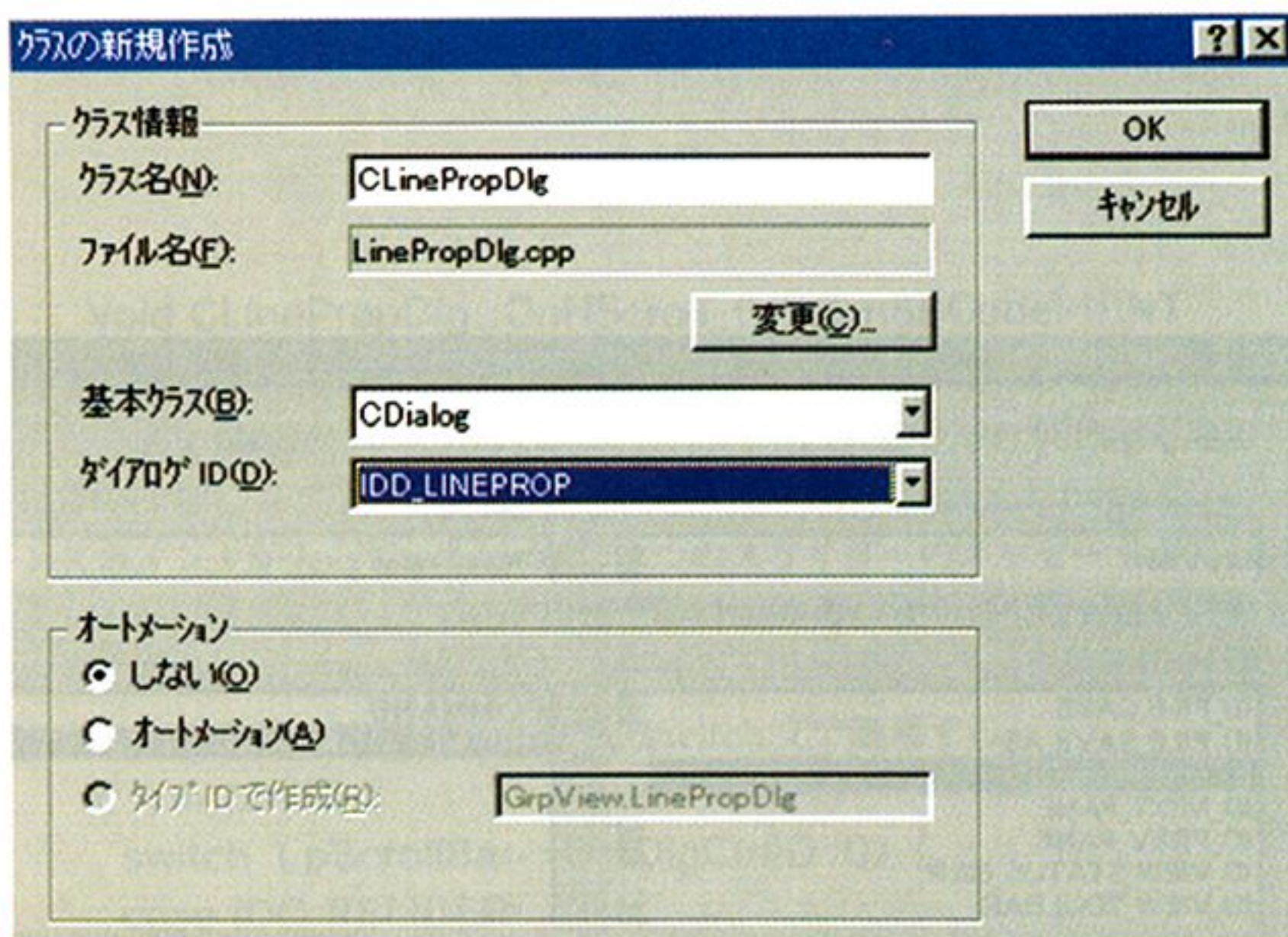



図24　clinepropdlg

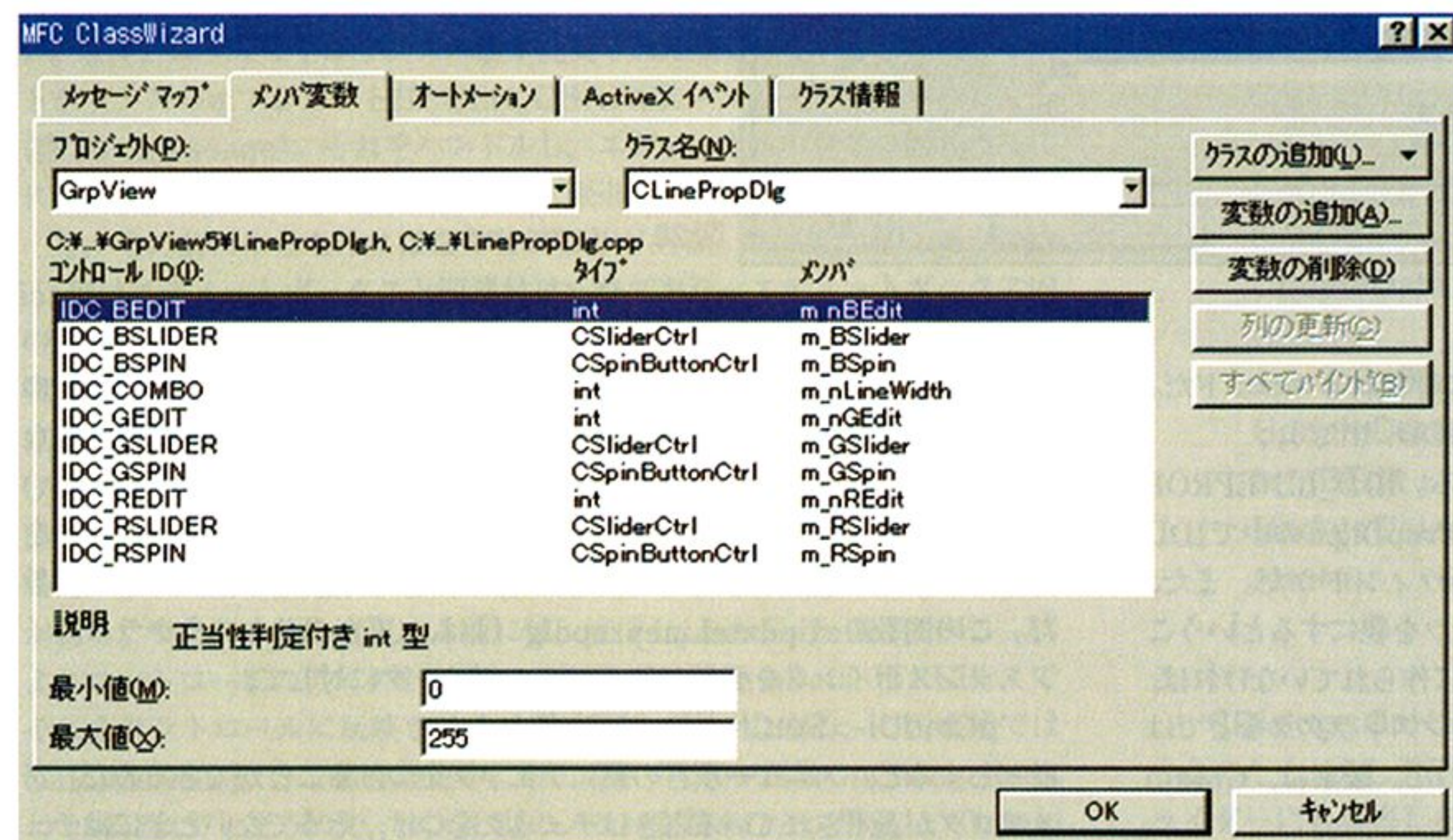



図25　member

からと，今回はツールバーにもボタンをつけてみよう。メニューは，IDR_MAINFRAMEとIDR_GRPVIETYPEという2つが作られている。MDIでは，子フレームが開いているときと，なにも開いていないときでは，別のメニューが使われるのだ。IDR_MAINFRAMEはなにも開いていないときのメニューで，IDR_GRPVIETYPEと比べると，[編集]と[ウィンドウ]のメニューが省略されている。ドキュメントが開いていない状態では必要ないからだ。このメニューの[表示]のいちばん下にでも，[ラインプロパティ]として追加しておこう。IDR_MAINFRAME，IDR_GRPVIETYPE両方ともだ(図26　menu)。

次はツールバー。こちらもリソースの中にIDR_MAINFRAMEというツールバーが作られているので，そこにボタンを追加する。まずいちばん右のなにも絵のないボタンをダブルクリックする。するとプロパティが現れるので，IDのリストメニューからさっきのメニューにつけたID_LINEPROPDLGを選択する。あとはボタンの表面の絵を描いてやるだけだ(図27　toolbar)。

これで準備はできた。それではコードを書いていこう。まずはメインフレームであるCMainFrameクラスに，CLinePropDlgクラスのメンバを追加する。publicにしておこう。

```
CLinePropDlg m_LinePropDlg;
```

次にCMainFrameのOnCreate ()でこのダイアログを作成する。ここ

```
void CLinePropDlg::OnMeasureItem (int nIDCtl,
            LPMEASUREITEMSTRUCT lpMeasureItemStruct)
```

nIDCtlというのはメッセージを発したコントロールのIDだ。これがコンボボックスIDC_COMBOであるときに処理すればよい。MEASUREITEMSTRUCTというのはアイテムの大きさなどを知らせる構造体で，コンボボックスの場合はこの中のitemHeightメンバに高さを設定すればよい。12ドットくらいでいいだろう。

```
if ( nIDCtl==IDC_COMBO ) lpMeasureItemStruct->
                              itemHeight = 12;
else CDialog::OnMeasureItem (nIDCtl,
                         lpMeasureItemStruct) ;
```

次はアイテム描画時のメッセージだ。

```
void CLinePropDlg::OnDrawItem (int nIDCtl,
            LPDRAWITEMSTRUCT lpDrawItemStruct)
```

DRAWITEMSTRUCTのうち，必要なメンバは次の3つだ。

itemID：
　これから描画するアイテムのID（0からのインデックス値）
hDC：
　アイテムのデバイスコンテキスト
rcItem：
　アイテムの領域を示す矩形

これらのメンバを使い，アイテムを描画していく。アイテムのIDに応じて太さが1から4までのペンを作り，それで適当な幅で線を引いてやればよい。設定した色を表示する領域を作るのを忘れていたので，ついでにここで色も表示してやろう。地が白なので，薄い色だと太さがわかりにくくなるが，それはご愛敬。

```
if ( nIDCtl==IDC_COMBO ) {
  UpdateData ( TRUE ) ;
  HPEN hPen = CreatePen ( PS_SOLID, lpDrawItemStruct
                                  ->itemID+1,
    RGB (m_nREdit,m_nGEdit,m_nBEdit)) ;
  HDC hDC = lpDrawItemStruct->hDC;
  HPEN hOldPen = (HPEN) SelectObject ( hDC, hPen ) ;
  RECT rect = lpDrawItemStruct->rcItem;
  MoveToEx ( hDC, rect.left+10, (rect.top+rect.bottom)
                                  /2, NULL ) ;
  LineTo ( hDC, rect.right-10, (rect.top+rect.bottom)
                                  /2 ) ;
  SelectObject ( hDC, hOldPen ) ;
  DeleteObject ( hPen ) ;
} else CDialog::OnDrawItem (nIDCtl, lpDrawItemStruct) ;
```

本来ならば，DRAWITEMSTRUCTのitemActionメンバを見て，フォーカスがあるときには破線で枠を描いたり，選択状態のときはリバースする必要があるのだが，面倒なので省略。自分でやってみてもらいたい。

ついに最後だ。やっとビュークラスに戻ってこれた。OnLButtonDown ()で，

```
CLinePropDlg * dlg = & ((CMainFrame *)
                     AfxGetMainWnd ()) -
>m_LinePropDlg;
CPen * pPen = new CPen ( PS_SOLID, dlg->
                          m_nLineWidth+1,
  RGB (dlg->m_nREdit,dlg->m_nGEdit,dlg->m_nBEdit)) ;
m_pOldPen = pDC->SelectObject ( pPen ) ;
```

としてペンを作ってデバイスコンテキストに選択し，OnLButtonUp ()で，

```
CPen * pPen = pDC->SelectObject ( m_pOldPen ) ;
delete pPen;
```

として廃棄すればいいだろう。描画中はペンを選択しっぱなしで構わないので，OnMouseMove ()は変更の必要はない。そうそう，ドキュメントに変更が加えられたことを示すSetModifiedFlag ()を忘れていたので，ついでにつけておく。

図31　GrpView-8

レジストリのアクセス

最後といったがもう少しだけ。せっかくラインにプロパティをつけられるようになったのだが，アプリケーションを一度終了してしまうと，プロパティは元に戻ってしまう。それは悲しいので，プロパティを保存するようにしよう。

Windows 3.1の頃は，こういった情報の保存にイニシャルファイル（拡張子INIのファイル）が使われていた。しかし，Windows 95からは，「レジストリ」というものに書き込むことを推奨している。これもファイルには違いないのだが，システムが管理する大きなデータベースといってよい。Windowsを使っている人なら耳にしたことくらいはあるだろう。これはアプリケーションのそういった情報だけでなく，システムの情報などもすべて含んでいる。そのため，むやみにこのレジストリが大きくなると，システムのパフォーマンスにも影響を及ぼす可能性があるので，レジストリを嫌ってあえてイニシャルファイルを使う人もいるようだ。しかし，レジストリにもメリットがある。たとえば保存する情報がわずかな場合，それでもひとつのファイルを作ってしまうので，HDDを1クラスタ消費する。数バイトのために8KBも16KBも無駄に使ってしまうのだ。FAT32ならばクラスタは小さいとはいえ，それでも4KB（8GBまで）を無駄にする。その結果，HDDのパフォーマンスの低下を招きかねない。その点レジストリならば，全体が大きなひとつのファイルであるから，クラスタギャップは最小限で済む。まあどっちもどっちだが，レジストリも綺麗に使っている限りはそれほどひどいことにはならないので，こちらを使うことにしよう。

レジストリの読み書きには，以下のようなCWinAppのメソッドが用意されている。

```
UINT GetProfileInt ( LPCTSTR lpszSection, LPCTSTR
                lpszEntry, int nDefault ) ;
CString GetProfileString ( LPCTSTR lpszSection, LPCTSTR
        lpszEntry, LPCTSTR lpszDefault = NULL ) ;
BOOL WriteProfileInt ( LPCTSTR lpszSection, LPCTSTR
                lpszEntry, int nValue ) ;
BOOL WriteProfileString ( LPCTSTR lpszSection, LPCTSTR
        lpszEntry, LPCTSTR lpszValue ) ;
```

メソッド名の先頭がGetのものが読み込み，Writeが書き込み，末尾がIntのものは数値でStringは文字列を扱う場合だ。いまはすべて数値なので，GetProfileInt ()とWriteProfileInt ()を使う。次のように記述すればよい。

```
m_nREdit = GetProfileInt ( "LineProperty", "Red", 0 ) ;
WriteProfileInt (."LineProperty", "Red", m_nREdit ) ;
```

GetProfileInt ()の第3引数は，情報がなかった場合のデフォルトの値だ。ではこれらを埋め込んでみる。読み込みはCLinePropDlgのOnInitDialog ()内でいいだろう。

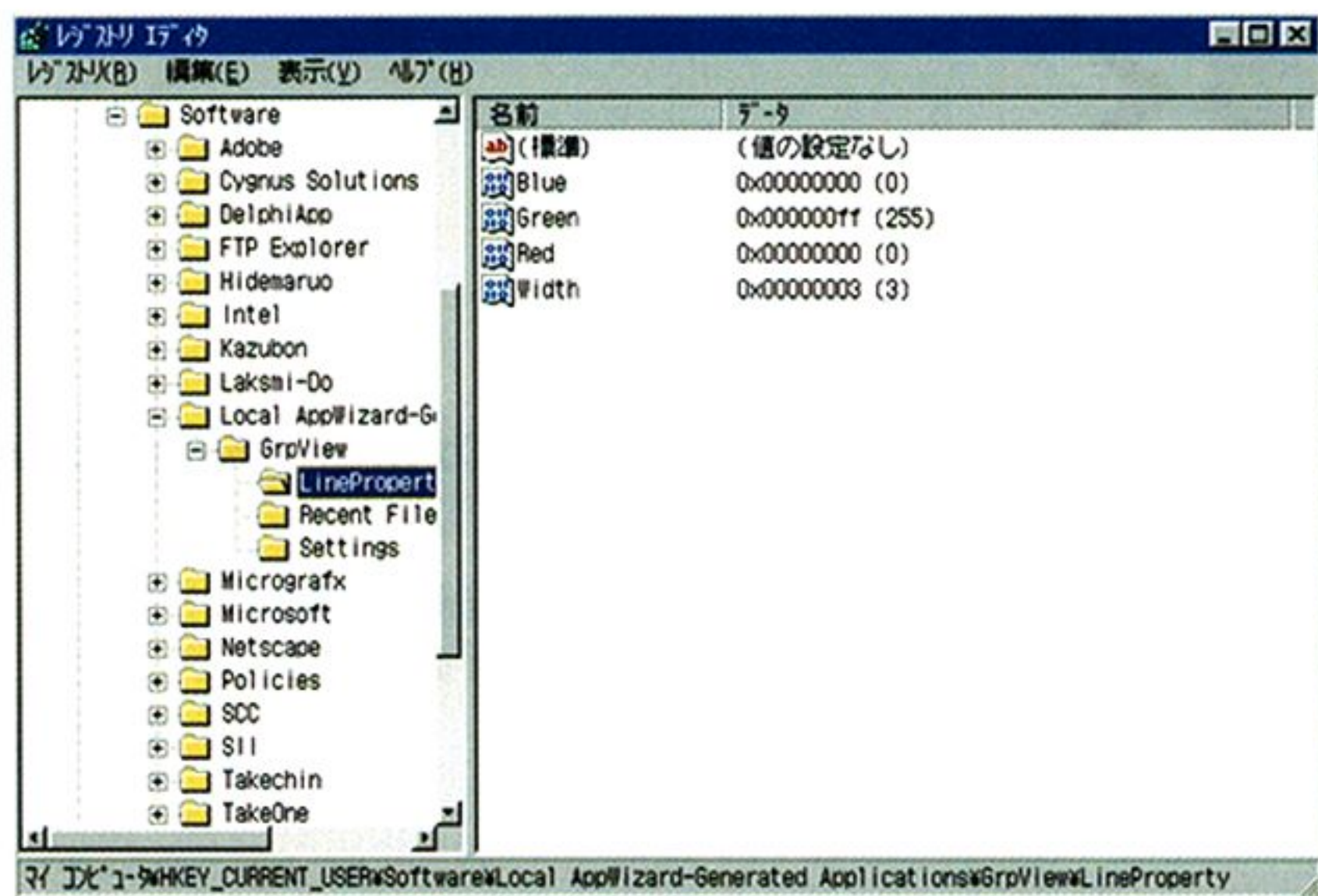



図32　REGEDIT

```
CWinApp * pApp = AfxGetApp () ;
m_nREdit = pApp->GetProfileInt ( "LineProperty", "Red",
                                                    0 ) ;
m_RSlider.SetPos ( m_nREdit ) ;
m_nGEdit = pApp->GetProfileInt ( "LineProperty",
                                          "Green", 0 ) ;
m_GSlider.SetPos ( m_nGEdit ) ;
m_nBEdit = pApp->GetProfileInt ( "LineProperty", "Blue",
                                                    0 ) ;
m_BSlider.SetPos ( m_nBEdit ) ;
m_nLineWidth = pApp->GetProfileInt ( "LineProperty",
                                          "Width", 0 ) ;
UpdateData ( FALSE ) ;
```

この辺は機械的にやればいい。

さて，書き込みだが，これはCLinePropDlgのDestroyWindow ()でもオーバーライドして，そこに記述することにしよう。このメソッドはウィンドウ(ダイアログ)が破棄されるときに呼ばれる関数だ。

```
CWinApp * pApp = AfxGetApp () ;
pApp->WriteProfileInt ( "LineProperty", "Red", m_nREdit
                                                    ) ;
pApp->WriteProfileInt ( "LineProperty", "Green",
                                          m_nGEdit ) ;
pApp->WriteProfileInt ( "LineProperty", "Blue", m_nBEdit
                                                    ) ;
pApp->WriteProfileInt ( "LineProperty", "Width",
                                          m_nLineWidth ) ;
return CDialog::DestroyWindow () ;
```

アプリケーションを普通に終了させた場合はこれだけで十分なのだが，もう少し問題が残っている。アプリケーションを終了させないまま，Windowsを終了させたときだ。その場合は，どうやらいちいちウィンドウを破棄するのを手抜きしているようで，このメソッドには処理が回ってこない。確実に処理が回ってくるのはCWinAppのExitInstance ()だが，普通に終了した場合には，すでにこのときダイアログが破棄されていて，ダイアログ内のデータを正常に取得することができない。なかなか難儀だ。そこで出番となるのがWM_ENDSESSIONというメッセージだ。これはWindowsが終了するときに送られるメッセージで，引数のBOOL値がTRUEのときはWindowsが終了することを示している(引数がFALSEなのはなにか意味があるのかっていうのは，ヘルプでWM_QUERYENDSESSIONでも見てほしい)。要は，このときにもLinePropDlgのDestroyWindow ()を呼んでやればいいわけだ。

```
if ( bEnding ) m_LinePropDlg.DestroyWindow () ;
```

これでオールジャスト完璧(サンプルGrpView6)。ラインのプロパティから選んだ線の色と太さで，自由に線が描けるはずだ(図31　GrpView-8)。また，いったん終了しても，ラインのプロパティが保存されていることも確認してもらいたい。

さて，ここではデータをレジストリに書き込んだわけだが，どんな感じで保存されているのかも知っておきたいところだ。レジストリの閲覧にはレジストリエディタというものを使う。これはWindowsに付属しているのだが，標準ではインストールされない。もし自分のパソコンにインストールされていなければ，[アプリケーションの追加と削除]でインストールしよう。また，インストールされても，スタートメニューには登録されない。Windowsディレクトリの中のREGEDIT.EXEというのがそれなので，自分でスタートメニューに登録するなりデスクトップにショートカットを作るなりしておこう。

そのレジストリエディタを起動すると，左側にツリービューが表示されており，[マイコンピュータ]の下にいくつかのフォルダ(キーという)がぶら下がっているはずだ。まずはその中の“HKEY_CURRENT_USER”というのを開いてもらいたい。すると，中からまたいくつかのキーが出てくるが，さらに“Software”，“Local AppWizard-Generated Applications”と辿ると，ほら出てきた，いま作った“GrpView”のキーだ。その中の“LineProperty”を見ると，確かに情報が書き込まれているのが確認できるだろう(図32　REGEDIT)。ところで気になるのは，“LineProperty”と同列にある“Recent File List”と“Settings”というキーだ。そんなキーは作った覚えがない。実はこれはMFCが勝手にやっていることで，“Recent File List”には，いわゆる[最近使ったファイル]のリストが記録されているのだ。“Settings”のほうは空っぽだが，なにか設定があったら書いてね，と提供されたキーなのだろう。

だいたいレジストリがどういったものかわかってもらえただろうか。WriteProfileInt ()などのメソッドで，第1引数と第2引数の文字列もどういった使われ方をするのかもわかったと思う。しかし，だ。“HKEY_CURRENT_USER”，“Software”まではいいとしよう。“GrpView”もアプリケーションの名前のキーだ。“Local AppWizard-Generated Applications”って長ったらしいのはなんなんだ？　AppWizardで作ったアプリケーションってのはわかるが，このままじゃカッコ悪すぎる。実はね，これはGrpView.cppの中，InitInstance ()中でスケルトンが吐き出していたのだ。

```
SetRegistryKey ( _T ("Local AppWizard-Generated
                                   Applications")) ;
```

この中を適当に自分の名前とかに書き換えればいい。そうすれば，仮に偶然にもほかの人が同じ名前のアプリケーションを作った場合でも，キーの衝突を避けることができる。では，このメソッドをもし呼ばなかったら(消してしまったら)どうなるか。実はその場合は，設定はレジストリではなく，Windowsディレクトリ中にイニシャルファイル，この場合ならGrpView.iniというファイルを作って，その中に保存されるのだ。つまり，MFCを使っていれば，このメソッドを呼ぶかどうかだけで，レジストリとイニシャルファイルの違いを吸収できてしまうのだ。どうしてもレジストリは嫌だという人は，いまできてしまったレジストリキーをレジストリエディタで削除して，上のメソッドも消してしまえばよい。ただし，誤って違うキーを消してしまうと，最悪Windowsが立ち上がらなくなる場合もあるので，気をつけよう。

大団円

そうか？　結局，画像を扱うったって，ダイアログにワンポイントで画像を貼り付けるとかいった程度ならともかく，真面目にグラフィックエディタとか作ろうと思ったら，この方法ではお話にならない，というのが期待していた方々の意見だろう。そりゃそうだ。やっぱデバイスコンテキスト越しじゃなく，直接ビット列にアクセスできなきゃね。幸い次回の特集はグラフィックらしい。この記事は別に特集にあわせる必要もないのだが，せっかくだから相乗りして，こんどはもっとディープに，「画像処理」といえるくらいまでコアな部分まで突っ込んでみたい。ついでにちょっとJPEGにも触れてみるとか。あくまでも予定だけどね。

C++ Builderで書くWindowsプログラム

岡田智亘　Okada Tomonobu

インプライズのC++ BuilderはGUIと高機能なコンポーネントベースのプログラミングシステムを組み合わせることで，非常に扱いやすいプログラミング環境を実現しています。ここでは，C++ Builderを使った開発の流れなどをまとめて掲載しておきましょう。

C++ Builderは簡単

Windowsでのプログラミング環境といえば，マイクロソフトのVisual Studioなどのシリーズが有名ですが，なにもそれでなくてはプログラムが作成できないというわけではありません。インプライズ(旧ボーランド)のDelphiは操作が簡単でハイパフォーマンスなアプリ制作環境として有名でした。ただ，Pascalベースのシステムなのであまり世の中には浸透していません。

DelphiのC++バージョンとして作られたのがC++ Builderです。わかりやすいプログラミング環境とパフォーマンスをそのまま受け継ぎ，コード記述部分だけはC++ベースで開発のできるようにされた環境です(実際，内部はDelphiと同じものらしい)。

C++ BuilderでWindowsプログラムを書くのはとても簡単です。それはC++ BuilderがDelphiやVisualBasicと同じくビジュアル開発環境だからです。フォームに部品，つまりコンポーネントを置き，それに対応する必要最小限のコードを書けばいいのです。「Cは書けるけどPascalを覚えるのもやだなーBASICもちょっとなー」という人にはC++ Builderがおすすめです。

現在の最新バージョンは4ですが，ver.3以降(以下，C++ Builder3)を使うようにしましょう。C++ Builder3とほかのC++との大きな違いはなんでしょうか。それはやはりクラスライブラリの違いでしょう。

OS上で動くプログラムを書く場合，OSに用意されている機能を使うためにシステムコールを呼び出します。WindowsではシステムコールはAPIといいます。32ビットアプリケーションの場合，Win32APIを使うことになります。APIは通常，低レベルの機能しか用意されていません。APIをいくつかまとめたり，C++で使いやすいように高機能な部品に変えたものがクラスライブラリだということができます。

C++ Builder3にはVCLというクラスライブラリが用意されています。Visual C++のクラスライブラリはMFCです。Visual C++でMFCを使う場合，MFCをある程度理解する必要がありますが，C++ BuilderでVCLを使う場合はビジュアル開発環境なのでそれほど深くVCLを理解する必要がないのです。

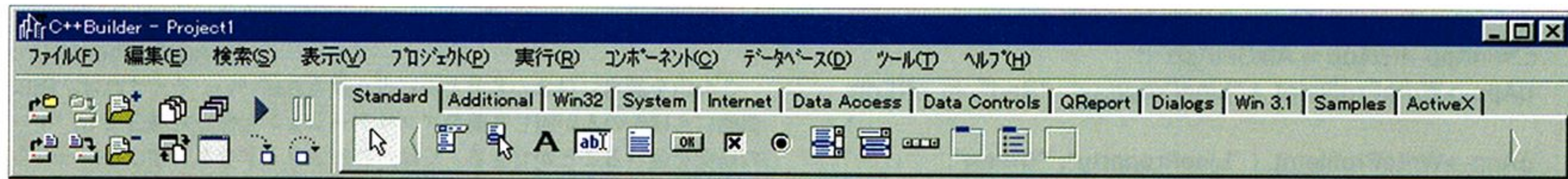



図1
メインウィンドウ

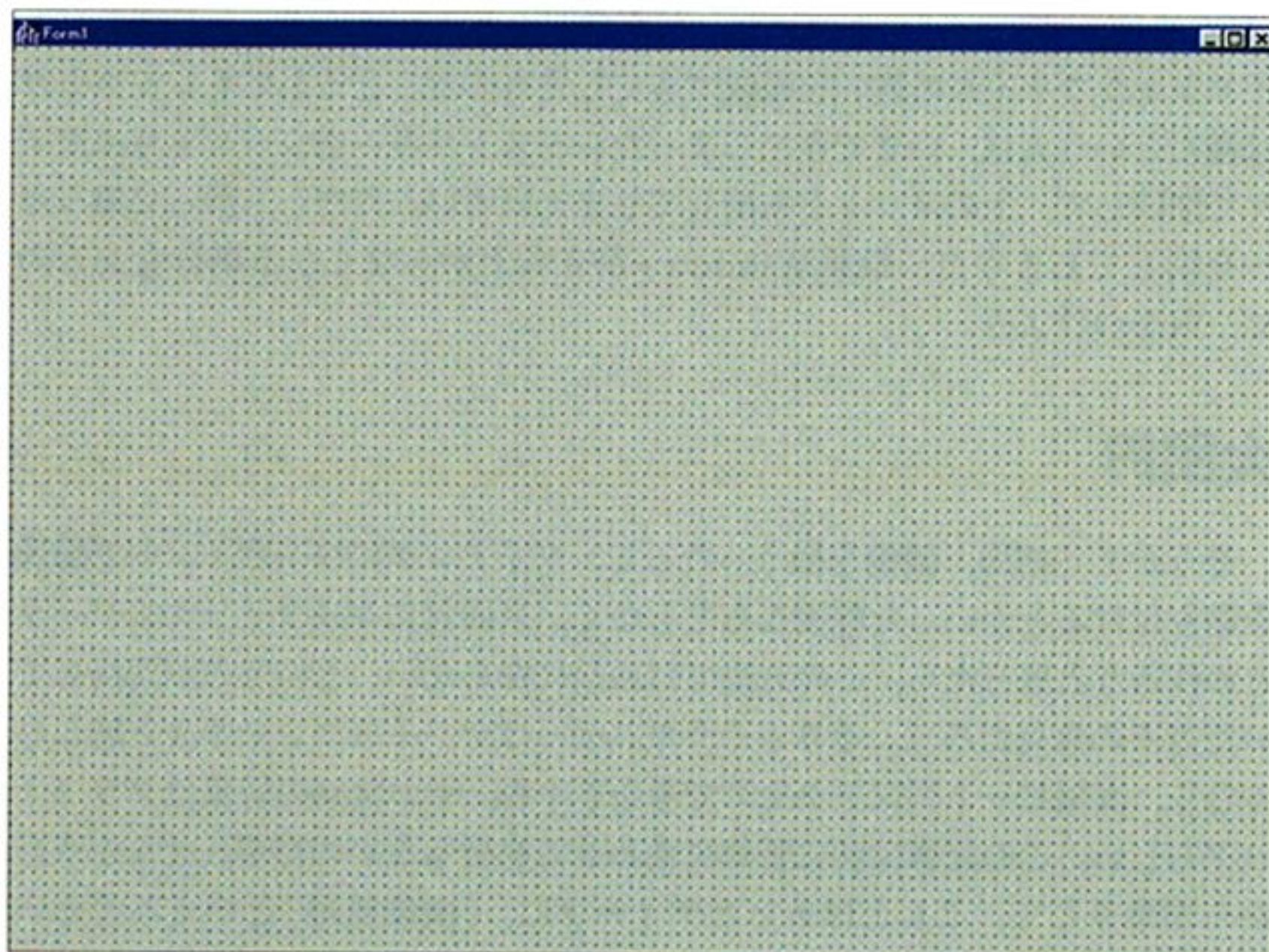



図2
フォーム

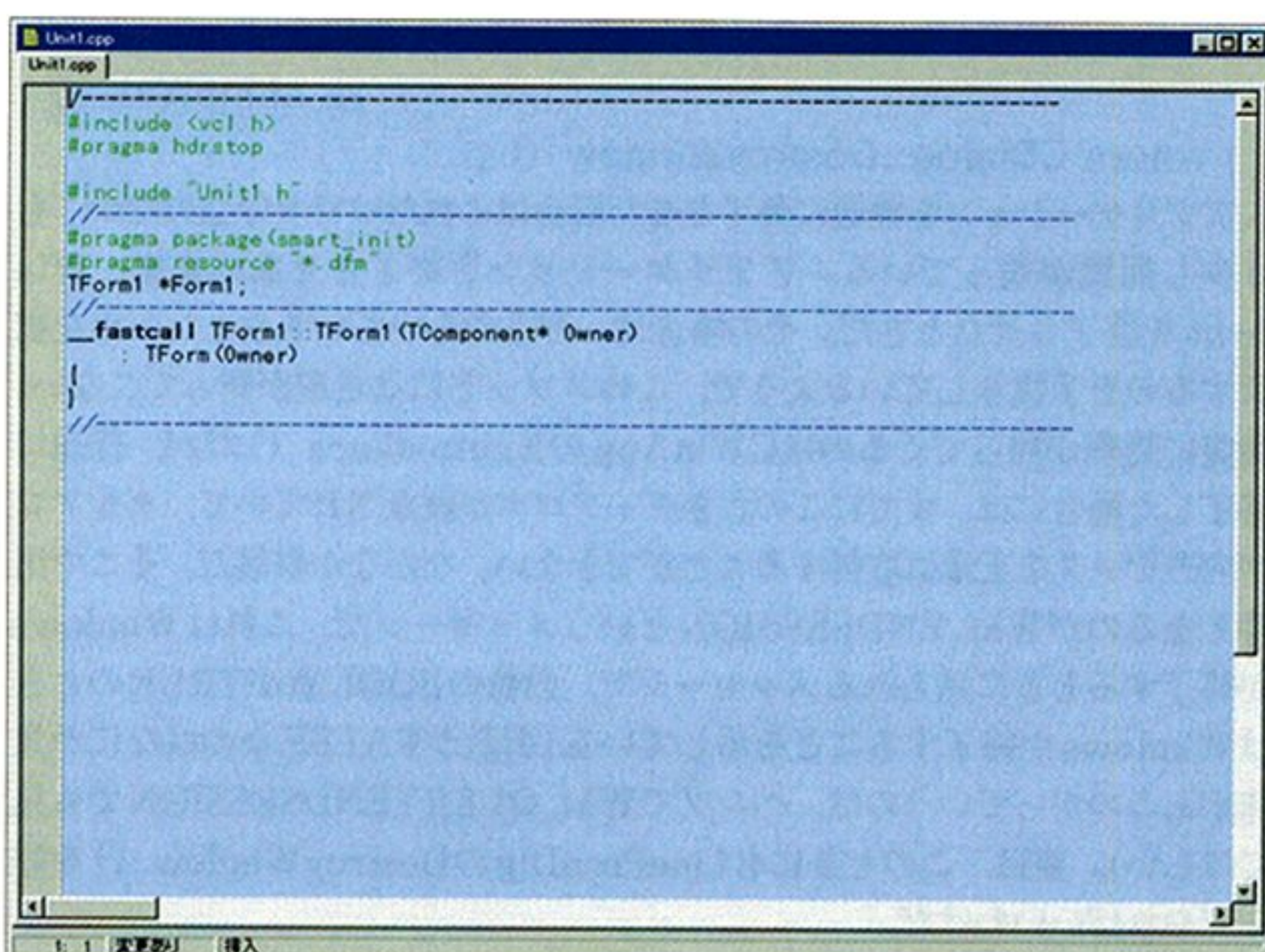



図3
コードエディタ

C++ BuilderとC++ Builder3の違い

最初に発売されたC++ Builderと現在のC++ Builder3以降のバージョンにはいくつかの違いがあります。そのひとつはプロジェクトメイクファイルの拡張子がmakからbprに変わったことです。C++ Builder用の入門書は多くありますがこれらの付録CD-ROMに入っているプログラムをC++ Builder3で使うにはプロジェクトメイクファイルをC++ Builder3のアイコンにドラッグ&ドロップするか，拡張子makをC++ Builder3に関連付けます。最初に実行したときプロジェクトメイクファイルの内容をC++ Builder3が自動的にC++ Builder3用に書き換えます。そのほかにはC++ Builder3にはプロジェクトグループの概念が加わったことです。プロジェクトグループは複数のプロジェクトを関連付けて管理することができます。

C++ Builder3を使ってみよう

C++ Builder3を実行するとメインウィンドウ，フォーム，コードエディタ，オブジェクトインスペクタが開きます。まず[ツール¦環境オプション]-[設定]-[自動保存の設定]-[エディタ]にはチェックをつけておきましょう。開発したプログラムにバグがあり，テスト時にフリーズしてC++ Builder3に制御が戻らなくなったときでも，この指定をしておけばソースコードをセーブして残してくれます。

プログラムを書くには[ファイル¦アプリケーションの新規作成]を選択しましょう。これでいままで開かれていたプロジェクトは終了し，新たなプロジェクトを開くことができます。このあとすぐ[ファイル¦プロジェクトに名前をつけて保存]を選択します。プロジェクト名とユニット名は好きな名前をつけまてかまいません。そうするとプロジェクトの名前がついたbpr，cpp，resの拡張子を持つ3つのファイルが作成されます。そのほかにユニット名のついたcpp，h，dfmの拡張子を持つファイルも作成されます。

[自動保存の設定]-[デスクトップ]にチェックがついている場合，プロジェクトを開き直したときやC++ Builder3を終了したときは，やはりプロジェクト名のついたdskの拡張子を持つファイルが作成されます。

.bprはプロジェクトメイクファイルです。

.resはリソースファイルでアイコンのイメージが入っています。

.dskはディスクトップ設定ファイルで開いているウィンドウの位置や大きさなどの作業状態が記録されています。

プロジェクト名のついた.cppはプロジェクトソースファイルです。ユニット名のついた.cppと.hはソースコードファイルとそのヘッダファイルでカスタム関数をここに書きます。.dfmにはフォームとそこに配置されたコンポーネントの情報が記録されています。

なお，メインウィンドウにはメインメニュー，ツールバー，コンポーネントパレットが並んでいますが，これらの環境はカスタマイズ可能です。好きな場所に好きな大きさで開いてかまいませんし，開かなくてもいいのです。ツールバーやコンポーネントパレットの内容も変えることができます。これらは毎回開かれるわけではなく[ツール¦環境オプション]-[設定]-[自動保存の設定]-[デスクトップ]にチェックがついている場合は前回終了してたときに開いていたものだけが開かれます。ただしメインウィンドウは必ず開かれます。メインウィンドウを閉じるとC++ Builder3も終了します。

図4
オブジェクトインスペクタ

図5
1行もコードを書かずに実行したプログラム

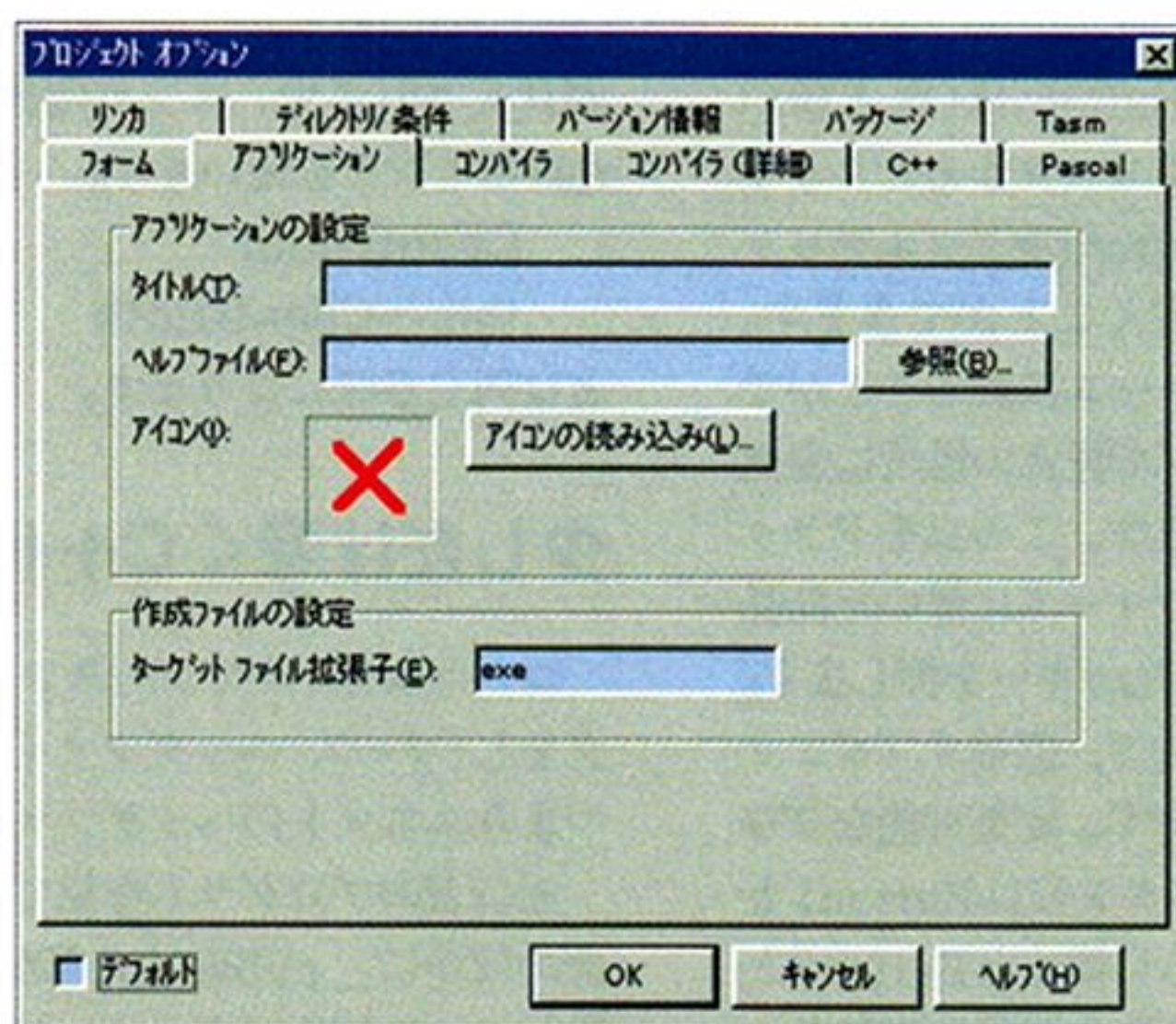



図6
プロジェクトオプションのウィンドウ

図7
クローズボタンの表示されたプログラム

ウィンドウを開く

[アプリケーションの新規作成]と[プロジェクトに名前をつけて保存]を行ったら，[実行]-[実行]を選択するかツールバーの[実行]を行ってみましょう。コンパイルをしてから実行されます。これでウィンドウを開くだけのプログラムは完成です。

1行のコードも書かずにできてしまいます。まあ，しかしこれだけではありがたみがありませんね，Visual C++でもこの程度は1行のコードも書かずにできます。さてこの状態では編集用のフォームが開かれたウィンドウの裏になって，フォームの編集作業ができません。それに実行させたままでもプログラムの編集作業はできるのですが，編集作業が終了したあと再度実行させるには現在実行中のプログラムを終了しなければいけません。

そこで実行中のプログラムを終了してしまいましょう。フォームの大きさを変えてみます。

フォームの四隅のどれかをドラッグして動かすとフォームの大きさを変えることができます。これだけです。

オブジェクトインスペクタでWidthとHeightもしくはClientWidthとClientHeightプロパティの値を変えることでもできます。Width，HeightとClientWidth，ClientHeightの違いはWidth，Heightはフォームの大きさそのもので，ClientWidth，ClientHeightはそれからタイトルバーとウィンドウの縁の大きさを引いた，フォームの使用可能領域の大きさです。タイトルバーに表示されるタイトルを変更するのはCaptionプロパティを変えることでできます。Positionプロパティはウィンドウの表示される位置と大きさを決めるためのものです。

この値はpoDefault，poDefaultPosOnly，poDefaultSizeOnly，poDesigned，poScreenCenterの5つの定数うちどれかを取ります。

poDefault
位置とサイズはWindowsが決める
poDefaultPosOnly
位置はWindowsが決めサイズは設計時のサイズになる
poDefaultSizeOnly
サイズはWindowsが決め位置は設計時の位置になる
poDesigned
位置とサイズは設計時のものとなる
poScreenCenter
サイズは設計時のサイズで位置は画面中央になる

この値をのpoDefaultPosOnlyにします。今度はアイコンを変更してみます。[ツール¦イメージエディタ]を選択しイメージエディタを立ち上げます。イメージエディタで[ファイル¦新規作成]－[アイコンファイル]を選択してアイコンを作成します。作成したアイコンをセーブしイメージエディタを終了します。[プロジェクト¦オプション]－[アプリケーション]－[アプリケーションの設定]の[アイコンの読み込み]ボタンを押します。

ここで先ほど作成したアイコンを指定します。イメージエディタで直接.resファイルを編集してもアイコンを変更することができます。ここで実行してみることにします。自分で設定したサイズ，タイトル，アイコンでウィンドウが開かれます。ここまで1行もコードを書いていません。プロパティを変更しただけです。

簡単でしょ。コンポーネントパレットの[Standard]の[Button]をクリックするとTButtonコンポーネントが選択されます。[Button]はOKと書いてあるボタンの絵のアイコンです。ここでフォームをクリックするとクリックした位置にButtonオブジェクトが置かれます。[Button]をダブルクリックすることでもフォームにButtonオブジェクトを置くことができます。Buttonオブジェクト以外の部品も同じやり方で，フォームに置くことができます。フォームに置かれた部品はドラッグ&ドロップで任意の場所に変更できます。部品の境界上に8個の黒い四角い点があります。この点をドラッグすることで部品の大きさを変えることができます。フォームに置いた部品を削除するときは部品をクリックして選択状態にしDeleteキーを押します。

これらはプロパティの値を変更することを意味するので，直接オブジェクトインスペクタでプロパティを変更することでも可能です。変更可能なプロパティは実行中のプログラムからも変更可能です。ボタンにはButton1という名前がついています。もうひとつボタンをフォームに置くとButton2という名前がつきます。このようにフォームに置いた部品の名前は部品名+数字となります。この名前は変更可能です。オブジェクトインスペクタのNameプロパティを変えればいいのです。

Nameプロパティは設計時のみ変更可能です。実行時にNameプロパティを変更した場合，その部品のプロパティやメソッドにアクセスしたときエラーになります。Captionプロパティを変更していないときにNameプロパティを変更すると，Captionプロパティも変更されますが，この2つは別々に設定することが可能です。ボタンに表示されている文字はCaptionプロパティの値になります。

ボタンのNameプロパティをCloseButton，CaptionプロパティをCloseに変更します。オブジェクトインスペクタの[イベント]ページに表示されている各イベントの右側をダブルクリックするとイベントに対応した関数がコードエディタのソースファイルに追加されます。フォームに配置された部品をダブルクリックすることでも同じことができますが，どのイベントになるかは部品ごとに設定されています。ボタンではOnClickイベントになります。

この関数はフォームのメンバ関数となります。この関数には中身がありません。ここに自分でコードを書きます。

ボタンをダブルクリックしOnClickイベントに対応するメンバ関数をコードエディタに追加します。ここで，

```
Close();
```

と書きます。このCloseはフォームのメソッドつまりメンバ関数で，フォームを閉じることができます。メインフォームを閉じると終了するのでこのボタンをクリックすることで終了することになります。このようなイベントに対応した関数をイベントハンドラといいます。

このボタンをフォーム中央に表示してみます。オブジェクトインスペクタで直接Top，Leftプロパティを変更することもできるのですが，プログラムで行うことにします。まずオブジェクトインスペクタのリストボックスでForm1を選択します。[イベント]ページでOnCreateをダブルクリックしコードエディタのソースファイルに関数を追加します。そこに次のコードを書きます。

```
CloseButton->Left = (ClientWidth - CloseButton->Width) / 2;
CloseButton->Top = (ClientHeight - CloseButton->Height) / 2;
```

フォームのOnCreateイベントはフォームが作成されたときに発生します。ClientWidth，ClientHeightはフォームのプロパティです。

これでボタンは中央に表示されます。

イベントハンドラを削除するのは中身だけを削除すれば，コンパイル時に自動的に削除されます。自分で全部削除してしまうとあとで面倒なので，必ず中身だけを削除するようにしましょう。

メンバ関数とはクラス名とスコープ解決演算子::でつながる関数です。

メンバ関数の内部ではクラスのprivateメンバでもprotectedメンバでもアクセスすることができます。クラスつまりオブジェクトのメンバにアクセスするための関数だということができます。もちろんそれ以外のこともできます。

メンバとそれを操作するメンバ関数をまとめることによって，privateメンバやprotectedメンバのようにクラスの内部構造を隠蔽化することができます。これをカプセル化といいます。

メンバ関数が呼び出されるとき，thisポインタにオブジェクトのアドレスが代入されます。メンバ関数はクラスのメンバにアクセスするとき暗黙でthisポインタを使います。

```
ClientWidth
this->ClientWidth
```

とは同じ意味になります。

少しだけ深くC++ Builder3を考えてみる

コードエディタでソースファイルのページが選択されているとき右クリックをし[ソース/ヘッダファイルを開く]を選択すると，そのソースファイルつまりユニットのヘッダファイルを開くことができます。

先ほどのプログラムを見てみると，クラスのメンバにprivate部，public部のほかに__published部があります。これはC++ Builder3が独自に拡張している部分です。コンポーネントパレットで選んだコンポーネントはここでメンバオブジェクトとして宣言されます。

メンバオブジェクトのイベントハンドラもここで宣言されます。ここはC++ Builder3が管理しているので，基本的には，ここに自分でなにかを書いてはいけません。ただオブジェクトを削除した場合，C++ Builder3は自動的にはイベントハンドラを削除しません。

このため自分でイベントハンドラの本体と宣言を削除しなければなりません。

メンバオブジェクトは残っているのにイベントハンドラを自分で削除してしまった場合は，このままコンパイルするとイベントハンドラの宣言は残っているのでコンパイルエラーが出ます。このため宣言も自分で削除しコンパイルします。そうするとC++ Builder3がメソッドへの参照を削除してもいいかと聞いてくるので，[はい]を選択します。これでやっと削除できたこと

になります。

C++ Builder3のクラスライブラリVCLはC++ではなくDelphiで書かれています。C++ Builder3はこれを使うためにC++を拡張してあります。

TControlクラスから派生したクラスはコントロールと呼びます。しかしTFormはTControlクラスからの派生クラスなのにコンポーネントです。コントロールはコンポーネントですが，コンポーネントは必ずしもコントロールではありません。コンポーネントかコントロールかは呼び方の違いだけなので気にする必要はありません。

すべてのコンポーネントはクラスです。オブジェクトはクラスのインスタンスです。つまりコンポーネントパレットに置かれている部品はクラスでフォームに置かれた部品はオブジェクトです。

これらのクラスの実体はVCLです。コンポーネントパレットにはC++ Builder3で書かれたコンポーネントも追加することができます。C++では新しいオブジェクトを生成したとき最初にコンストラクタが呼ばれるのですが，C++ Builder3では__published部に書かれたVCLオブジェクトは，コンストラクタが呼ばれるより先に生成されています。どこにもnew演算子を使ってオブジェクトを生成する記述がないにもかかわらずです。

このためコンストラクタの中でVCLオブジェクトのプロパティを変更したりメソッドを呼び出すことができます。

プロパティという概念はC++にはありません。これもVCLを使うためにC++ Builder3が独自の拡張をしています。プロパティはクラスのメンバ変数，メンバオブジェクトのどちらかです。プロパティ値の代入と読み出しにデータメンバを直接アクセスするだけではなく，アクセスメソッドつまりメンバ関数の指定もできます。VCLではイベントはプロパティの一種です。

グラフィックを使ってみる

ウィンドウズのプログラムを書いてみようと思ったとき，どのようなプログラムを書いてみたいでしょうか。やはりグラフィックを使ったプログラムではないのでしょうか。ぼくはそうでした。

ゲームを作るにしてもグラフィックを使います。C++ Builder3にはグラフィックを扱うクラスがいくつかあります。フォームにグラフィックイメージを表示するためにTImageコンポーネントかTPaintBoxコンポーネントを使うかCanvasプロパティを使うのですが，ここではTImageコンポーネントを使うことにします。TImgaeコンポーネントはコンポーネントパレットの[Additional]にあります。TImageコンポーネントはメンバオブジェクトとしてCanvasオブジェクトを含みます。CanvasオブジェクトはTCanvasクラスのオブジェクトです。

VCLではクラス名にはTypeの略としてのTが頭につき，このTをとったものがメンバオブジェクトのオブジェクト名になります。TCanvasクラスはWindowsのGDI(Graphic Device Interface)をカプセル化したものです。

TImageコンポーネントはメンバオブジェクトとしてTPictureクラスを持っています。TPictureクラスはTBitmapクラスをメンバオブジェクトとして持っていて，TBitmapクラスはTCanvasクラスをメンバオブジェクトとして持っています。

図8
Sample1の画面

```
Image1->Canvas
Image1->Picture->Bitmap->Canvas
```

この2つは同じことを意味します。

これはVCLではメンバオブジェクトはポインタとして宣言されTImageクラスのCanvasとPicture->Bitmap->Canvasが同じオブジェクトを指すからです。

しかしImage1->Canvasと指定してメソッドかプロパティをアクセスする以前に，Image1->Picture->Bitmap->Canvasと指定してメソッドかプロパティをアクセスしても表示は更新されません。最初にImage1->Canvasと指定してメソッドかプロパティを使ったあとならば表示は更新されます。

ビットマップイメージを保持するのにTBitmapクラスを使用するのですが，TBitmap型のオブジェクトを宣言するときにスコープ解決演算子を使いGraphics::TBitmapとして宣言します。これはWindowsのAPIの.BITMAP構造体を再定義したWindows::TBitmapと区別するためです。これを行わないとどちらのTBitmapかわからずコンパイル時にエラーになります。

背景とキャラクターを重ね合わせるにはTImageListコンポーネントを使うのが楽です。TImageListコンポーネントは同じサイズのイメージを収納します。イメージと一緒にイメージのマスクも収納することができるので，擬似スプライトとして扱うことができます。

背景とキャラクターを表示したら，やはり動かしたくなります。キーボードの入力にあわせて動かすためにフォームのOnKeyDownイベントを使います。

OnKeyDownイベントはキー入力があると1回発生します。なぜOnKeyPressイベントではないのかというとOnKeyPressイベントは押されたキーのASCII文字を登録するためカーソルキーなどの判定には使えないからです。

ゲームではフォームの大きさを変えないことが多いと思います。このプログラムでも変えません。

フォームのBorderStyleプロパティをbsSingleに変えます。BorderStyleプロパティの値は，

bsDialog　サイズ変更不可。標準のダイアログボックス境界
bsSingle　サイズ変更不可。一重境界線
bsNone　サイズ変更不可。可視境界線なし
bsSizeable　標準のサイズ変更可能境界
bsToolWindow　bsSingleと同じだがキャプションが小さい
bsSizeToolWin　bsSizeableと同じだがキャプションが小さい

の6つの定数のどれかになります。

同じ理由で最大化を禁止します。これにはBorderIconsプロパティのbiMaximizeの値をfalseにします。

Positionプロパティの値をpoDefaultPosOnlyに変えます。このプログラムがSample1です。実行するためには実行ファイルと同じディレクトリの中にBitmap001.bmpとBitmap002.bmpが必要です。Bitmap002.bmpには背景，Bitmap001.bmpにはキャラクターです。このキャラクターはDOGA-L1で作りました。カーソルキーで上下左右に動きます。Sample1では2つ以上のキー入力を同時に認識できません。それにキーボードと本体の間に接続するタイプのジョイパッドで動作に問題があるものもあります。

キーを押しっぱなしにした場合，キーボードからの入力はリピートがかかるのにジョイパッド側ではかからないタイプのものです。これを解決するためにOnKeyUpイベントとOnTimerイベントを使います。OnTimerイベントを使うためTTimerコンポーネントを追加します。

TTimerコンポーネントはコンポーネントパレットの[System]にある時計のアイコンです。キーの状態を保持するフラグを用意し，このフラグをOnKeyDownイベントでtrueにしOnKeyUpイベントでfalseにします。キャラクターの移動や表示の処理はOnTimerイベントのイベントハンドラで行います。Timer1のIntervalプロパティを20に変えます。これはOnTimerイベントが発生する間隔でミリ秒単位です。

このプログラムがSample2です。Sample2の実行にもSample1と同じ2つのbmpファイルが必要です。Sample2では2つのキーを同時に認識するので斜め移動も可能です。

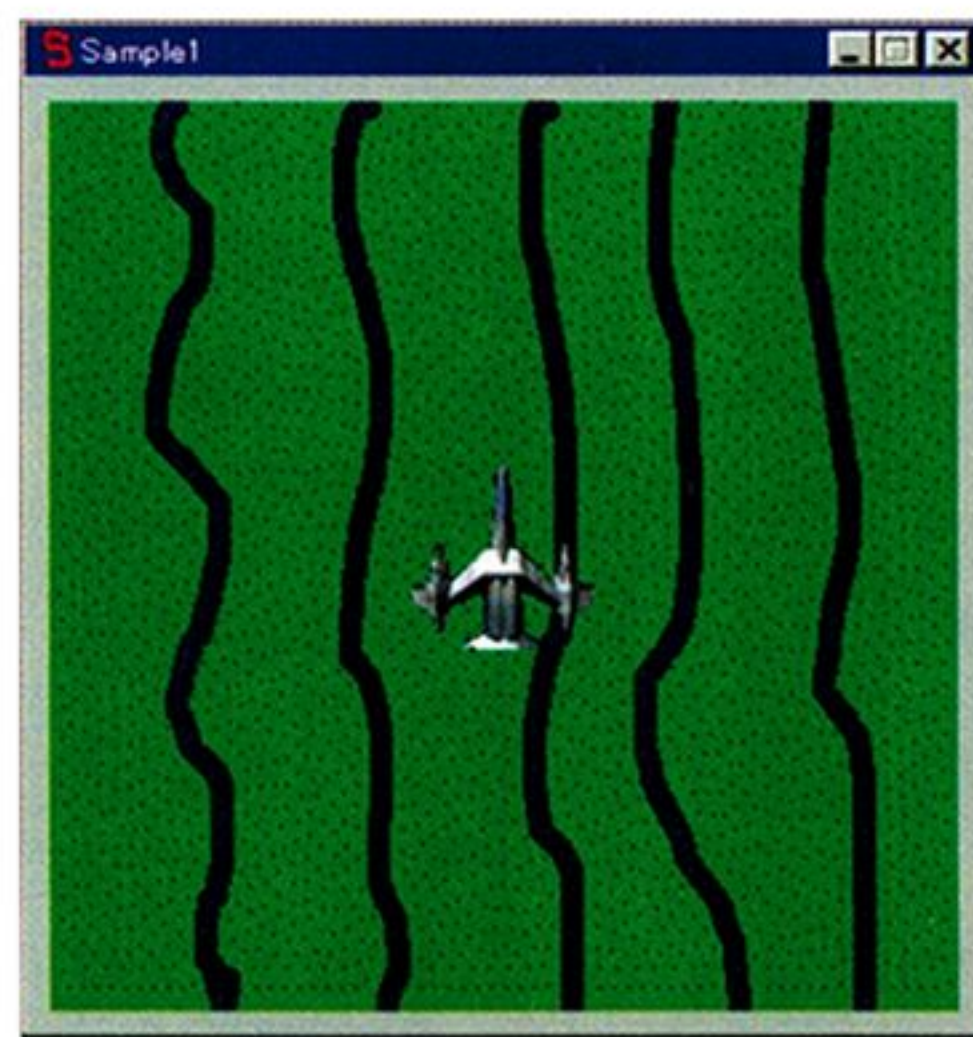



図9
Sample3の画面

当たり判定処理を作る

今度は敵も表示して当たり判定を作ってみましょう。当たり判定にはいろいろな方法がありますが，今回はマスクイメージを使います(編注：ドット単位でのイメージの重なりを判定する方法ですが，シューティングゲームなどでは必ずしもベストの方法ではありません)。マスクイメージはImageListのマスクイメージを使わず，独自に作成することにします。まず，このためのメソッドを作りましょう。

```
void TForm1::CreateCharaJudgeData(Graphics::TBitmap
 * Bitmap,
   unsigned char * JudgeData)
{
   int i, j, k, c;
   TColor MaskColor;

   MaskColor = Bitmap->Canvas->Pixels[0][0];

   for(i = 0, c = 0;i < CHARAHEIGHT;i++)
      for(j = 0;j < CHARAMW;j++, c++)
         for(k = 0;k < MWIDTH;k++)
            JudgeData[c] = JudgeData[c] |
               (unsigned char)((Bitmap->Canvas->Pixels[j
                * MWIDTH + k][i]
               != MaskColor) << k);
}
```

これがマスクイメージを作成するメソッドです。CHARAHEIGHT = 64，CHARAMW = MWIDTH = 8です。Bitmapにはキャラクターのイメージを指定します。このキャラクターの大きさは64×64ピクセルです。

JudgeDataには作成したマスクイメージを収納するためのunsigned char型の配列を指定します。Bitmapの左上のピクセルを透過色として収納します。物体が0の場合存在せず1の場合存在することになります。1バイト=8ビットなのでunsigned char型ひとつに8ピクセル分収納することになります。

メンバ関数ではなくグローバル関数でもかまわないのですが，キャラクターの大きさが固定になっているなど汎用性が低いのでメンバ関数にしてみました。

当たり判定用のメソッドも作成します。

```
bool TForm1::JudgeContact(unsigned char * JudgeData1,
int x1, int y1, unsigned char * JudgeData2, int x2, int y2)
{
int xs, xc, yc, w, h, Judgex, Judgey, ContactFlag = 0;
bool f;

Judgex = x1 - x2;
Judgey = y1 - y2;

if (-CHARAMW < Judgex && Judgex < CHARAMW &&
   -CHARAMH < Judgey && Judgey < CHARAMH) {
   if (Judgey >= 0) {
      yc = 0;
      h =(CHARAMH - Judgey) * MHEIGHT;
   } else {
      yc =-Judgey * MHEIGHT;
      h = CHARAHEIGHT;
   }
   if (Judgex >= 0){
      xs = 0;
      w = CHARAMW - Judgex;
   } else {
      xs =-Judgex;
      w = CHARAMW;
   }
   for (;yc < h;yc++)
      for (xc = xs;xc < w;xc++)
         ContactFlag = ContactFlag | (JudgeData1[yc *
CHARAMW + xc] &
            JudgeData2[(yc + Judgey * MHEIGHT) *
CHARAMW + xc + Judgex]);
}

if (ContactFlag)
   f = true;
else
   f = false;

return f;
}
```

これが当たり判定用のメソッドです。CHARAMH = 8です。

JudgeData1とJudgeData2にはマスクデータを指定します。x1，y1，x2，y2には座標を指定します。この座標はピクセル単位ではなく仮想座標です。キャラクターの移動が8ピクセルずつなのでピクセル座標/8で指定します。

このため横方向の当たり判定をunsigned char型の配列の1要素ずつ行っています。JudgeData1の要素とその座標に対応するJudgeData2の要素のビットごとのANDをとります。これを重なり合っているすべての要素に対して行い，そのすべての値のビットごとのORをとります。この値はContactFlagに入っています。ContactFlagが0以外だと物体が重なり合っているということです。高速化のために最初にボックス判定で各要素が重なり合っているかを判定します。

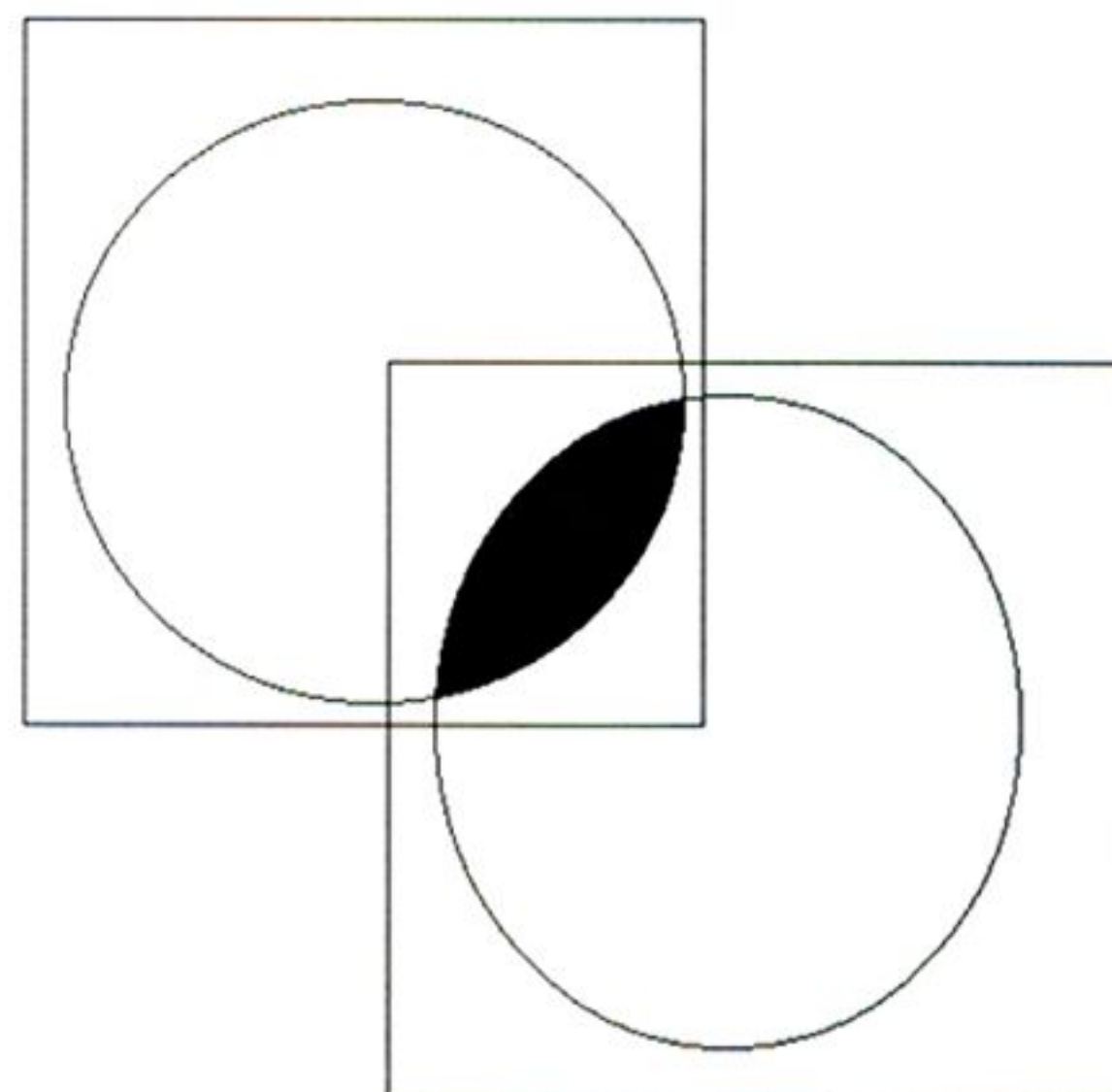



図10
当たり判定のイメージ

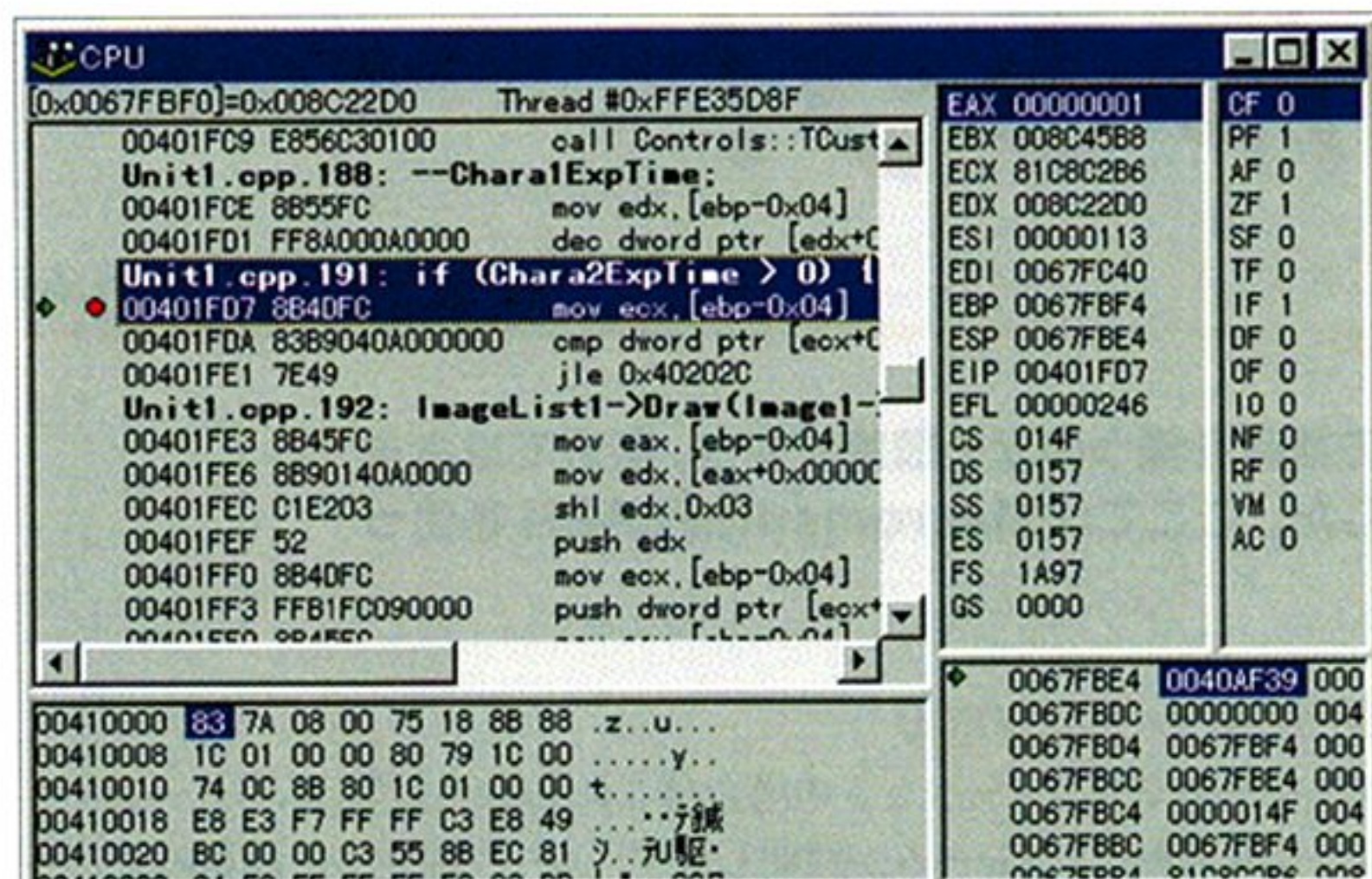



図11
CPUウィンドウ

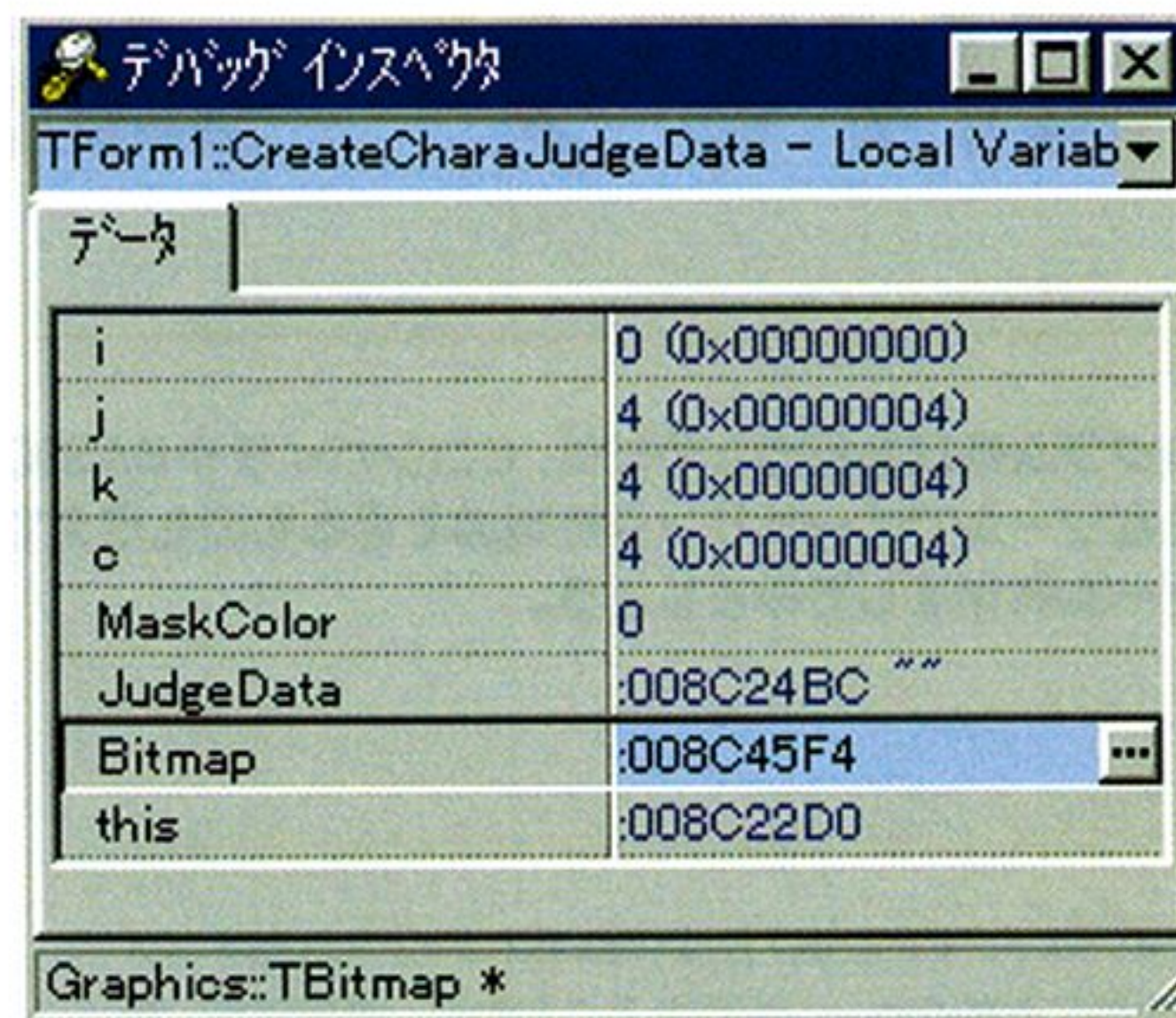



図12
ローカル変数のインスペクタウィンドウ

弾を撃つ

Zキーを押すと弾を撃つようにします。連射ではなく1回押すと1発発射するようにします。このために2つのフラグを使います。

2つのフラグが00のとき発射可能です。キーが押されたとき10になりこのとき発射します。発射したあと11になります。キーを放すと01になり，そのあと00になります。このプログラムがSample3です。Sample3の実行には同じディレクトリの中にCBitmap.bmpが必要です。

オフスクリーンビットマップを使う

画面に大量のイメージを描画するなど時間のかかる処理をすると，その過程も画面に表示されてしまいます。これを避けるためにもう1枚ビットマップを用意しておき，イメージの描画はこのビットマップに対して行います。すべてのイメージの描画が終わったときに一気にこのビットマップのイメージを表示します。これをオフスクリーンビットマップといいます。ゲームでは一般的に行われている画面切り替えですね。

しかし，TImageコンポーネントに対してこれを行おうとするととても遅くなってしまいます。これはTImageコンポーネントも内部にビットマップを保持していて，これがオフスクリーンビットマップの役目を果たすため，自分で用意したオフスクリーンビットマップが無意味になってしまい，なおかつイメージの転送を2回も行うため遅くなってしまうのです。

TImageコンポーネントの表示を更新するタイミングを制御するのは難しいので，代わりにTPaintBoxコンポーネントを使うことにします。これはコンポーネントパレットの[System]にあります。このプログラムがSample4です。Sample4にもSample3と同じbmpファイルが必要です。

デバッグ作業をする

ある程度大きいプログラムが一度で動くことはまれです。ありえないといってもよいでしょう。思ったように動作しない原因を調べなければいけません。そのためには調べたいところにブレークポイントを設定しステップ実行かトレース実行を行います。

ブレークポイントを設定して実行するとブレークポイントの場所まで実行しそこで停止します。そこからトレース実行もしくはステップ実行を行います。別のブレークポイントを設定し，再度そこまで実行させたほうが便利な場合もあります。

トレース実行またはステップ実行は，1回にC++の1行または機械語の1命令ずつを実行します。トレース実行とステップ実行の違いは，トレース実行では，ほかの関数が呼び出されると実行ポイントはその関数に移動するけどステップ実行では移動しないことです。

トレース実行でデバッグセッションを開始した場合，もしくはCPUウィンドウにフォーカスがある場合は1命令ずつの実行になります。CPUウィンドウは[表示¦CPU]で開きます。

ローカル変数の値を見るためには[実行¦ローカル変数のインスペクト]を選択します。それ以外の値は[表示¦監視式]を使うのが便利です。監視式を追加するためにはコードエディタで範囲を指定し右クリックをし[この単語を監視式に追加]を選択します。実行中ではないと[監視できません]と表示されますが気にすることはありません。実行をしてその変数もしくは式の中で使われている変数がスコープの範囲に入れば値が表示されます。これらの方法では実行がいちいち停止するので使いにくいときがあります。一瞬で値が切り替わる変数には使えない方法ですが，タイトルバーに見たい変数の値を表示すると便利な場合があります。

リリース

自分で使うだけならこのままでよいのですが，配布したり，配布しようとする場合には問題があります。それはランタイムライブラリを使っているからです。作成したプログラムだけを配布しても，そこにランタイムライブラリがインストールされていなければ，プログラムは実行できません。このためランタイムライブラリも一緒に配布するか，ランタイムライブラリを使わないようにしなければなりません。

ここではランタイムライブラリを使わない方法を説明します。

[プロジェクト¦オプション]を選択しプロジェクオプションのウィンドウを開きます。[パッケージ]-[実行時パッケージ]-[実行時パッケージを使って構築]と[リンカ]-[リンク]-[ダイナミックRTLを使用]のチェックをはずします。RTLはCのランタイムライブラリです。実行時パッケージはVCLのランタイムライブラリです。これでプログラム単体で配布できるようになります。

最適化も行いましょう[コンパイラ]-[スピード設定]-[リリース]を選択します。これでコンパイルを行いましょう。少し小さくなったはずでし，少し速くなったはずです。劇的変化ではないのですが，気分の問題です。おまじないのつもりくらいで行いましょう。

終わりに

VCLだけでもたいていのことはできるのですが，VCLでは実現できないことやより高速に行いたいときには，Win32APIやDirectXを直接使わなければなりません。今回はWin32APIの使い方を説明しなかったのですが使うのは簡単です。DirectXもマイクロソフト製品以外の開発環境での使用を考慮するようになってきています。

C++ Builder3の開発能力はとても高度です。自分で小さいツールなどちょっとしたものを作成しようと思ったときにはVisual C++などより簡単に使えてしまいます。やはりビジュアル開発環境は便利です。これまでWindows環境を敬遠していた人も，自分でなにか作ってみてはいかがでしょうか。

FLASHでゲームを作ろう!

伊藤のりゆき　Ito Noriyuki

プログラム処理を行うためには，Cコンパイラや BASICなどの言語を使うのが一般的ですが，最近ではさまざまなツールでオーサリング処理のできるものも増えてきました。ここではMacromediaFLASHを使ってゲーム制作をしてみましょう。

FLASHとは

Macromedia FLASH(以下，FLASH)とはWeb用に開発されたアニメーションツールです。手軽に動的なページが作れることから，登場以来，爆発的に普及してきました。その特徴は次のとおりです。

1) ベクターデータによる描画のため，拡大してもジャギーのない綺麗な出力が可能
2) データサイズが，JPEGやGIFと比べて小さいため，Webに最適
3) スクリプティングなしに，ボタンなどのインタラクティブ機能を作成可能

現在のバージョン3になって，特にインタラクティブ面が強化され，アイキャッチなどのWebアニメーションにとどまらず，ショートゲームの制作も可能となりました。今回は，そのFLASHを使ったゲーム制作を紹介いたします。

FLASHの基礎

FLASHは，本来アニメーションツールです。そのため，時間軸であるタイムライン(図1)を基本に動きます。タイムラインは細かくフレーム単位で区切られ，1秒間に数フレームを連続表示して，パラパラマンガの仕組みでアニメーションを行います。

たとえば，「人が歩く」というアニメーションを作りたいとします。このとき，フレームごとに少し動作を変化させた絵を描くだけで，目的のアニメーションを制作することは可能です。しかし，FLASHでは人物の腕，足，胴体，頭などと，オブジェクト単位でデータをライブラリ化し，フレームごとに位置を変えて，アニメーションをします。そのほうがデータ効率がよくなるからです。

頭，腕，胴体，足などのライブラリ化されたオブジェクトを，FLASHでは「シンボル」と呼びます。シンボルにできるものは(FLASH内で描いた)グラフィックから，あるひと固まりのアニメーション，ボタンなど，以下のように3種類あります。

図1
FLASH起動時の画面と各部の名称

(1)グラフィックシンボル

グラフィックやアニメーションの場合はこれを使います。ムービーの1フレームとシンボルの1フレームが同期します。ループ設定や，開始のフレーム番号の指定などが可能です。

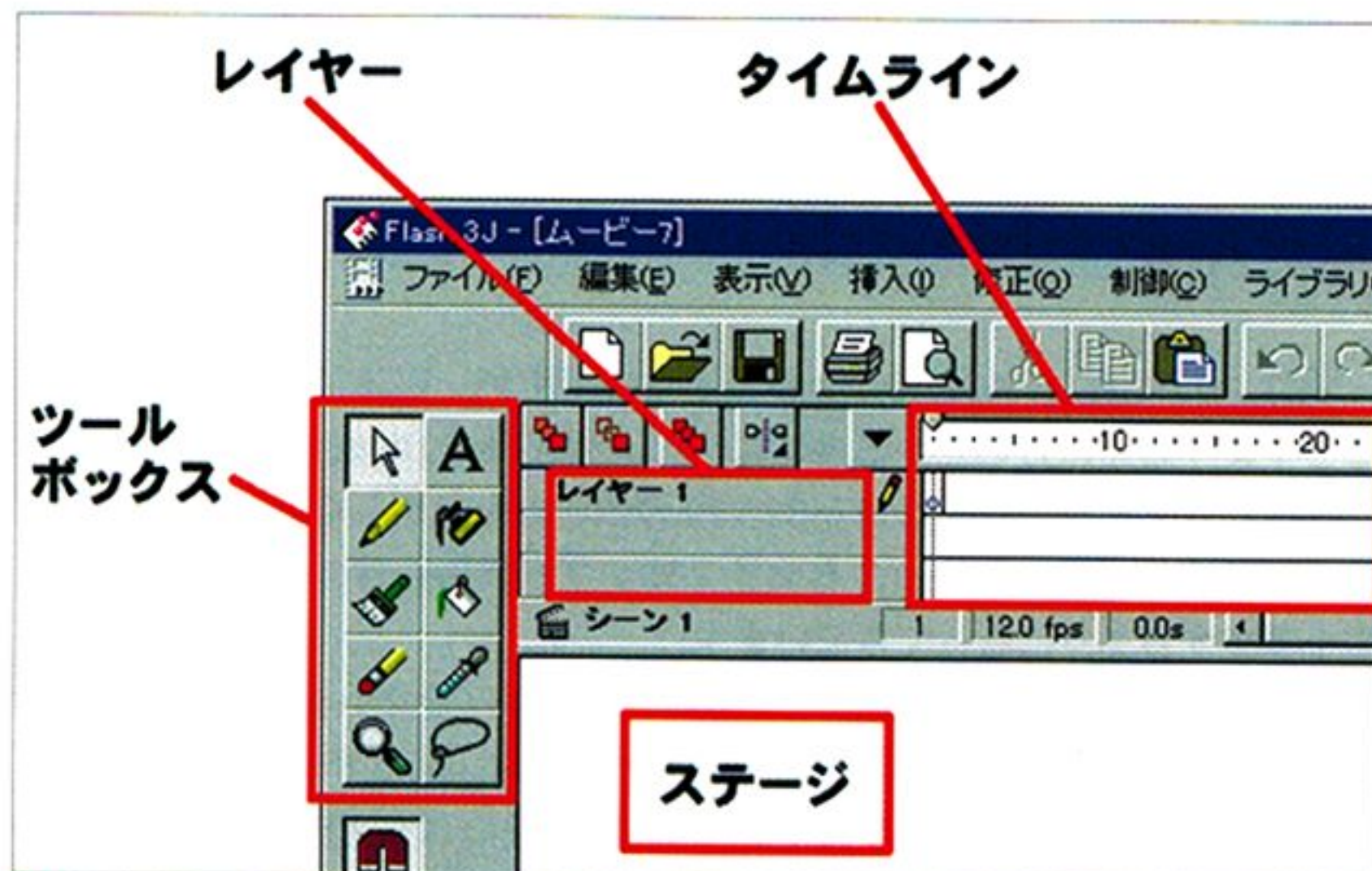
(2)ボタンシンボル

ボタン専用です。シンボルのタイプをボタンに設定するだけで，スクリプティングなしにボタンを制作することが可能です。

表1　マウスイベント一覧

イベント	説明
Press/マウスダウン	オブジェクト上にポインタがあるときにマウスボタンを押すとアクションが発生します
Release/マウスアップ	オブジェクト上にポインタがあるときにマウスボタンを離すとアクションが発生します
Release Outside/領域外でマウスアップ	オブジェクトの領域外にポインタがあるときにマウスボタンを離すとアクションが発生します
Roll Over/ロールオーバー	ポインタがオブジェクトに接したときにアクションが発生します
Roll Out/ロールアウト	ポインタがオブジェクトの領域外に出たときにアクションが発生します
Drag Over/ドラッグオーバー	オブジェクトの領域外から内側にマウスをドラッグしたときにアクションが発生します
Drag Out/ドラッグアウト	オブジェクトの内側から外にマウスをドラッグしたときにアクションが発生します

表2　アクション一覧

アクション	説明
Go To	指定したシーンおよびフレームに移動し，オプションでムービーを再生します
Play	カレントフレームからムービーを再生します
Stop	カレントフレームでムービーを停止します
Toggle High Quality	アンチエイリアス表示をオンまたはオフに切り替えます
Stop All Sounds	すべてのサウンドを停止します
Get URL	URL から，指定したウィンドウにURLで指定したドキュメントを読み込みます
FS Command	JavaScriptの呼び出しを行うときなどに使用します
Load Movie	指定したレベルに，URL からムービーを読み込みます
Unload Movie	Load Movie で読み込んだムービーを指定したレベルから削除します
Tell Target	ターゲットシンボルインスタンスにアクションを送ります。「ターゲット」には，アクションを送りたいムービークリップシンボルのインスタンス名を指定します
If Frame is Loaded	指定したフレームが読み込まれると，If ステートメントに続くアクションが実行されます
On MouseEvent	指定したイベントが発生すると，On 条件に続くアクションが実行されます。「イベント」で選択できるオプションは，Press(マウスダウン)，Release(マウスアップ)，Release Outside(領域外でマウスアップ)，Roll Over(ロールオーバー)，Roll Out(ロールアウト)，Drag Over(ドラッグオーバー)，Drag Out(ドラッグアウト)です

(3)ムービークリップ

内容はグラフィックシンボルと同じですが，大きな違いがあります。それは，ムービーとの同期を取らないところです。そのため，メインムービーのタイムラインから独立して再生することが可能です。また，後述する「Tell Target」に不可欠なシンボルです。

FLASHでスクリプティング?!

FLASHはWebアニメーションツールと紹介しましたが，バージョン3からスクリプティングに近い機能を持つようになりました。それが「Tell Target」と呼ばれるものです。

「Tell Target」は，FLASH内のほかのシンボルの制御を可能にします。たとえば，人が歩くムービークリップに対して，「Stop」と命令を送れば，歩いていたムービークリップはアニメーションがストップします。また，「Play」と命令を送れば，ムービークリップはアニメーションを再開します。

この「Tell Target」をそれぞれ，「Play」「Stop」ボタンに組み込めば，簡単にインタラクティブなムービーを制作できます。

FLASHでは，「Tell Target」に代表されるコマンドを，「アクション」と呼びます。アクションは，マウスイベントやフレーム通過をトリガとして動作します（表1「マウスイベント一覧」，表2「アクション一覧」）。

Tell Targetの基礎

簡単な「Tell Target」を作成してみましょう。例は，数字のムービーの再生・停止を2個のボタンから行うものです（図2）。

1)ムービークリップを作る

まず，数字の切り替わるムービークリップを作りましょう。メニュー「挿入＞シンボルの作成」でシンボルプロパティ画面が表示されます。名前を「numbers」とつけて，「ムービークリップ」にチェックを入れ「OK」ボタンを押します。まっさらなステージと，1フレームだけのタイムラインが現れます。

中心にテキストツールで「1」と文字を打ちます（ステージ中央の「+」が，このシンボルの中心です）。次にタイムラインの2フレーム目を選択して，右クリックし，「フレームを挿入」を選択します（図3）。すると，フレームが2フレーム目まで伸びます。2フレーム目を選択し，もう一度，右クリック，「キーフレームを挿入」します。2フレーム目にマーカーがあるのを確認して，先ほどのテキストをダブルクリックし，テキストを「2」に変更します。

メニュー「制御＞再生」を実行してみましょう（マーカーがループ再生しないときは，メニュー「制御＞ループ再生」にチェックを入れてから再生します）。1フレーム目と2フレーム目の文字がチカチカとアニメーションしています。

2文字では情けないので，10フレームまでフレームを挿入して，キーフレームを作り，それぞれ，3から9，0を入力しておきましょう。

2)ボタンを作成

次に，数字ムービーを制御するためのボタンを作成します。まず1)と同じ手順でシンボルを作成します。メニュー「挿入＞シンボルの作成」のあと，シンボル名を「play」，「ボタン」を選択して「OK」を押します（図4）。

今度は，「numbers」とは違った大きなタイムラインが現れます。これは，「play」シンボルがボタンの機能を持っていることを示します。タイムライン上には「アップ」「オーバー」「ダウン」「ヒット」と4つのフレームがありま

図2
サンプル

図3 フレームを挿入
右クリックでポップアップメニューを出します。Macの場合は，「control＋クリック」です

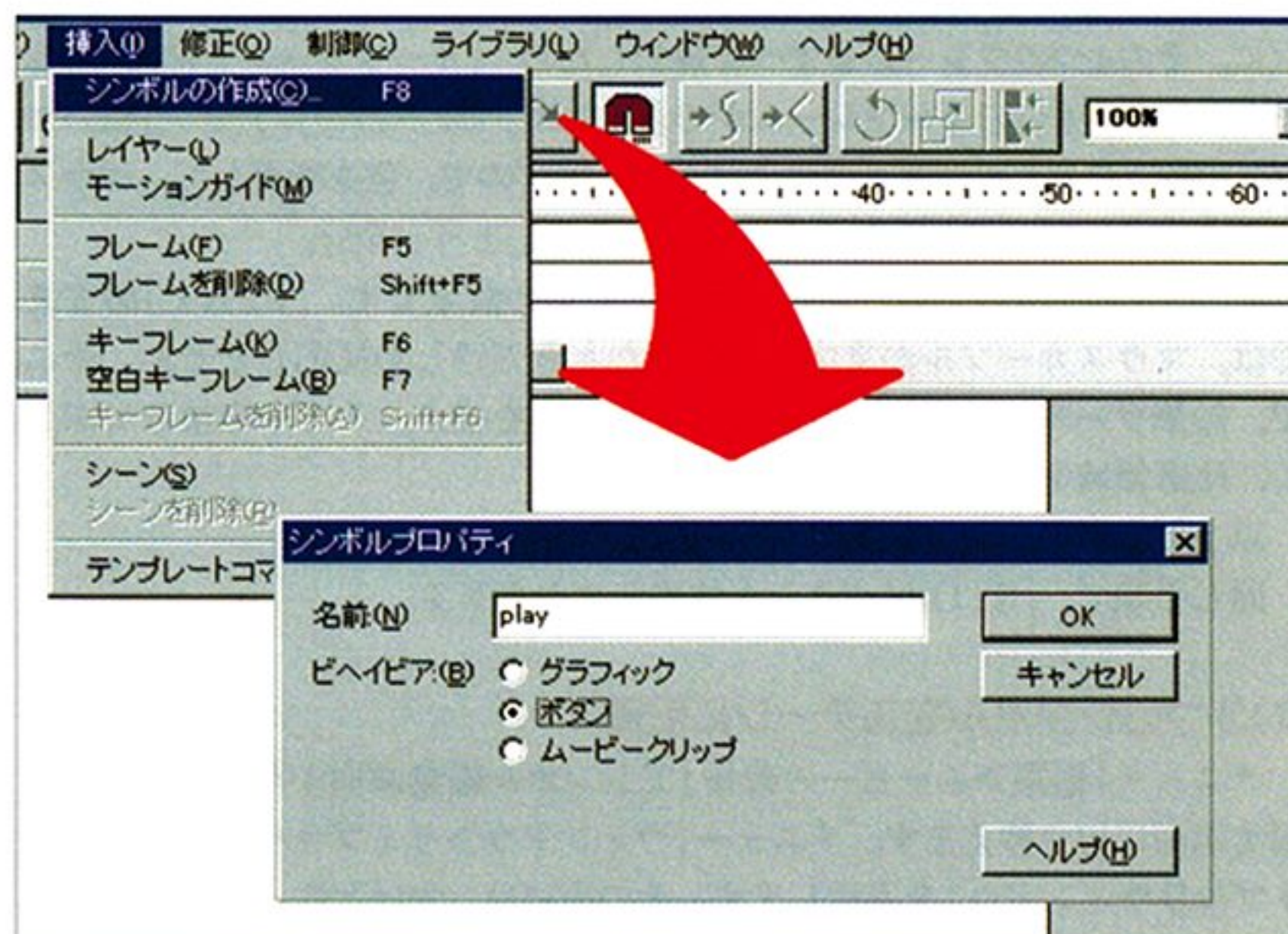



図4
ボタンの作成

図5
ボタンを構成する4つのフレーム

図6　ボタンのオーバー
カラーパレットから選び，オブジェクトの色を変更します

図7　ボタンのヒット
「ヒット」の四角を塗る色は，何色でも関係ありません。また，テキスト「PLAY」は必要ありませんので，削除します

す(図5)。

これら4つのフレームは，「アップ」がなにもイベントの起こらない状態，「オーバー」はマウスカーソルがシンボルの上に重なった状態，「ダウン」がクリックした状態を示し，「ヒット」がこれらマウスイベントの反応領域を表します。

テキストツールを使って「PLAY」と文字を打ちます。さらに「ヒット」のフレームを選択し，「フレームを挿入」をして，4つのフレームを埋めます。次に，その4つのフレームをすべて選択したら，「キーフレームを挿入」をします。これで4つのキーフレームができました。「オーバー」「ダウン」は，マウスの動きにあわせて表示されるフレームですので，色を変更して，マウスの動きがユーザーにわかるようにしておきましょう(図6)。

「ヒット」のフレームは，ボタンの反応領域の設定です。「PLAY」のままでは，マウスカーソルが文字の上にきたときだけしか反応しません。そこで，鉛筆ツールでテキストを囲むように四角を描き，その中を塗りつぶして，反応領域を広げます(図7)。

以上で，ボタンシンボルが作成できました。

同じ手順で，「STOP」ボタンも作成しておきます。

3)3つのシンボルをステージに配置

メニュー「編集>ムービーの編集」でシンボル編集画面から，ムービーの編集画面に切り替えます。メニュー「ウィンドウ>ライブラリ」を使い，「ライブラリウィンドウ」を表示します。その中には，先ほど作った「シンボル」が収められています。「numbers」「play」「stop」という名前のシンボルをひとつずつステージにドラッグします(図8)。なお，ドラッグできるのは，サムネイル表示されているグラフィックだけです。これで，好きな場所にシンボルを配置します。

4)インスタンス名を設定

ムービークリップ「numbers」は「Tell Target」される側です。「インスタンス名」とは，「Tell Target」する際に目標となるターゲットの名称で，必ず設定しなくてはなりません。「Tell Target」からは，このインスタンス名に対して，アクションを送ります。そのため，インスタンス名のないムービークリップは「Tell Target」で制御できません。また，名称を間違えている場合も同様に機能しません。

ステージ上の「numbers」シンボルをダブルクリックします。「インスタンスプロパティウィンドウ」が開きます。ビヘイビアの項目が[ムービークリップ]になっているのを確認して，[インスタンス名]を「numbers_clip」と定義します(図9)。

インスタンス名は特に規則がありませんが，ムービー内では一意になるようにして，ムービークリップの内容を見わけやすい名称にするのがよいでしょう。

5)ボタンにアクション「Tell Target」を設定

ステージ上のボタン「PLAY」をダブルクリックすると，4)と同じように「インスタンスプロパティウィンドウ」が表示されます。[ビヘイビア]が[ボタン]になっていることを確認します。ボタンの「インスタンスプロパティ」には，[アクション]タブが追加されていますので，そのタブを選択します。

「+」のボタンが，アクションの追加ボタンです。クリックすると，追加で

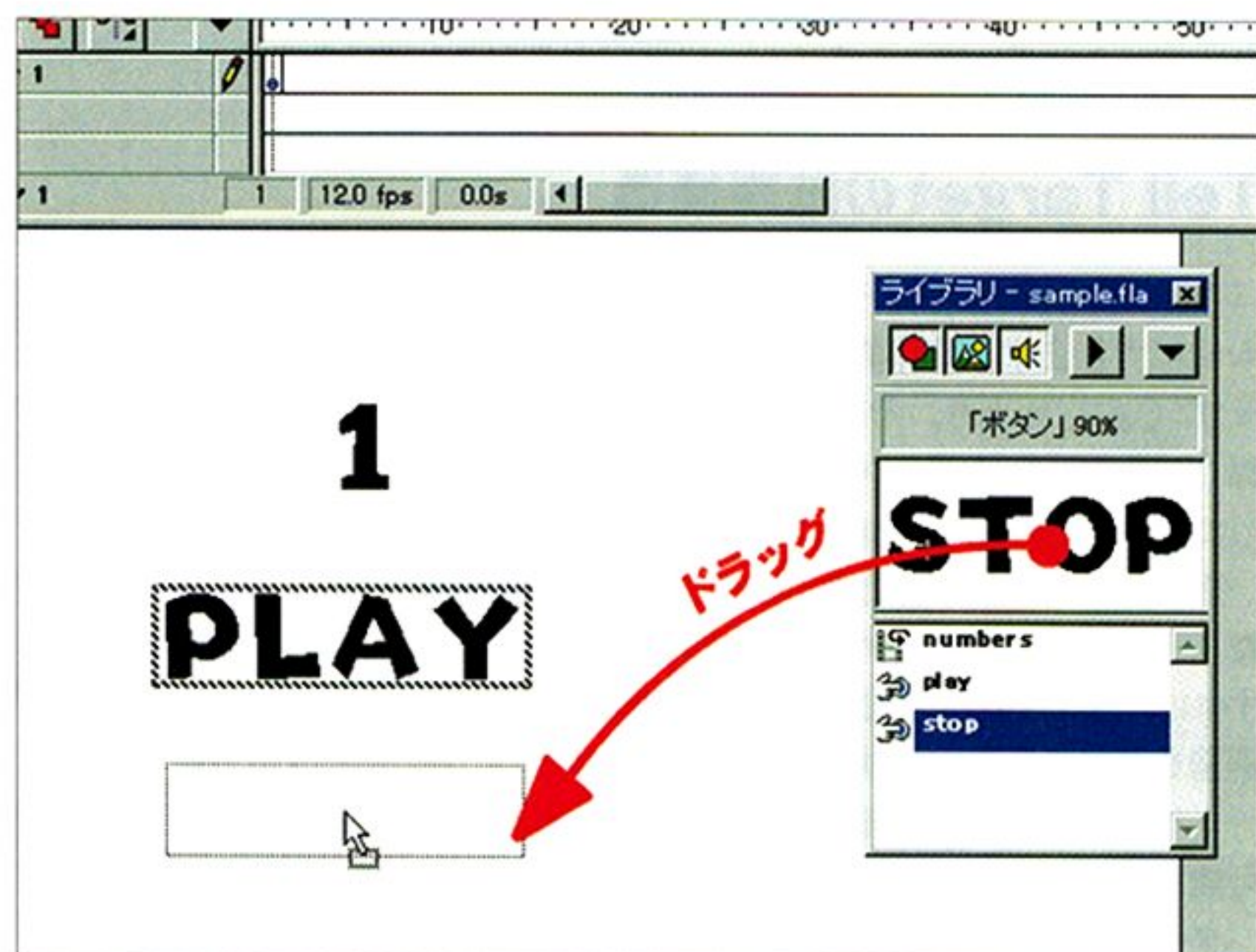



図8　シンボルの配置
ライブラリウィンドウからドラッグ＆ドロップで配置します

図9　インスタンスプロパティウィンドウ
ステージ上のシンボルをダブルクリックすると開きます。インスタンス名は赤い枠の中のとおりに設定します

図10　アクションの設定
「＋」ボタンを押し，メニューのなかからアクションを選び，インスタンス名の一覧から，目的のターゲットをダブルクリック，または，テキスト入力して，ターゲット名を入力します

図11　ムービーのテスト
ムービーの再生テストには「ムービープレビュー」を実行します。ファイルの転送をグラフ化して見ることもできます

きるアクションのリストが出てきます。このなかから［Tell Target］を選択します。すると，［パラメータ：Begin Tell Target］のところに，4）で設定したインスタンス名が現れます。そのインスタンス名をダブルクリックして，ターゲットに指定します（図10）。

これで，このボタンからの命令がターゲットの指すムービークリップに伝えられるようになりました。あとは，アクションの具体的内容を追加していきます。アクションの［Begin Tell Target］を選択した状態で，もう一度「＋」ボタンを押し，［Play］アクションを選択します。これで，「PLAY」ボタンが定義できました。次に「STOP」ボタンを定義します。これも「PLAY」と同様に定義をします。異なるのは，［Play］アクションでなく，［Stop］アクションだという点です。

6）ムービーのテスト

これで，すべての定義が終わりました。さっそくテストしてみます。ムービークリップを使っている場合，テストは，実際にShockwaveFlashファイル（SWF）を作成して行う必要がありますが，SWF作成と再生を同時に行うメニュー「制御＞ムービープレビュー」を実行します（図11）。数字が自動でカウントアップされています。次に「STOP」ボタンを押してみましょう。数字のカウントアップが止まりました。「PLAY」ボタンを押すと，また，再生が始まります。

7）応用１

数字のムービーは自動で最初からアニメーションしていました。これは，ムービー本体のタイムラインとは独立してループ再生されるという，ムービークリップの性質のためです。もし，ムービークリップを静止の状態で始めたいという場合は，ムービークリップの1フレーム目に［Stop］アクションを入れておきます。

また，ループ再生は［Stop］アクションのない2フレーム目にジャンプするように定義し直します（ジャンプするだけでなく，再生もさせる必要がありますので，［Go to and Play］のチェックを忘れずにします）。なぜなら，1フレーム目の［Stop］アクションで止まってしまい，ループ再生できないからです。

修正したら「ムービープレビュー」で確認します。数字ムービーが最初から止まっているようになりましたが，ループしている数字の中に「1」と「0」がありません。「1」が表示されないのは，先ほど，2フレーム目へジャンプするように設定したためです。そこで，1フレーム目を選択して，「フレームの挿入」を実行します。すると，1フレーム目がひとつ後ろにずれ，2フレーム目も「1」になります。これで，ループするムービー中に「1」が表示されるようになりました。

「0」が表示されない理由は，「0」のフレームにアクションで，「Go To」を書いたためです。アクションは，フレームへ入る直前に処理されます。そのため，実際には「0」を表示する前に，2フレーム目へジャンプしているのです。そこで，「0」のある11フレーム目も延長し，アクションのあるフレームは，キーフレームの直前で「フレームの挿入」を行い，アクションを最後のフレームに移動します。

図12　リストが空白
[Tell Target]の[ターゲット]リストや，[Go To]の[ラベル]リストは，ときどきなにも出てこないことがあります。リストに出せるのは同一階層のものだけだからです。ラベル名が正しければ，正常に機能しますので，直接入力しておきましょう

8）応用2

「PLAY」ボタンで数字ムービーは続きから再生を始めますが，強制的に「1」から再生したいという場合などは，「PLAY」ボタンに指定のフレームへジャンプする機能をつけることができます。これは，「応用1」で説明したとおりです。このとき，フレームに「ラベル名」を設定して，ラベル名にジャンプすることが可能です。ラベルを使うと，フレームの絶対番号がわからないときや，フレームの挿入・削除などで，フレームの長さが変わる可能性のあるときなどに便利です。

ラベルの設定は，[フレームプロパティ]の[ラベル]で行います（フレームをダブルクリックすると出てきます）。あとは，[Go To]アクションの[ラベル]で，ラベル名の候補リストから，選ぶだけです。条件により，候補リストに指定したいラベル名がないことがあります。ラベル名が正しければ，正常に機能しますので，直接入力しましょう（図12）。

9）応用3

ボタンのデフォルトのイベントは「Release/マウスアップ」です。そこで，「PLAY」ボタンをロールオーバーで機能するように変更してみましょう。「PLAY」ボタンのアクションの「On (Release)」を選択します。パラメータには，マウスイベントのチェックボックスが並んでいます。「Release/マウスアップ」のチェックを外し，「Roll Over/ロールオーバー」にチェックを入れます。これで，ロールオーバーに反応するボタンができます。

図13　ヘルメッ豚98（licensed by KaZuhiro FuRuhata）
古籏さんの名作ゲーム「ヘルメッ豚」のWindows 98版（本当か？）

Tell Targetの階層構造

「Tell Target」のターゲットになるムービークリップはネストした状態でも使用できます。具体的な例でいうと，右から左へ動く車のムービークリップ「CAR」があり，その中のタイヤもムービークリップ「TIRE」として登録してあるとします。つまり，車のシンボルの中に，タイヤのシンボルが入っている状態です（シンボルは何重にでもネストできます）。

車の動きを制御するときは，いままでと同様にターゲットは「/CAR」となります。さらに車のタイヤだけを制御したいときは，「/CAR/TIRE」とすれば，タイヤだけを制御することが可能です。ターゲットの指定は，階層構造をそのまま表します。タイヤのムービークリップ（子）から車のムービークリップ（親）を制御するときには，「../CAR」といった相対パス表現も可能です（ファイルシステムのパス表現と同じです）。

ターゲットの指定は，相対パスだけでなく，絶対パスも可能です。そのとき，フラッシュのムービー（ステージ）をルートとしてとらえ，「_flash0」から始める書き方をします。

すべてのターゲットは，相対パスと絶対パスのどちらでも表現可能なので，わかりやすい方法を取るとよいでしょう。

ムービークリップ変数

FLASHでは変数などは用意されていません。当然，ゲームなどを作る場合には，スコアのカウントを取ったり，自機の残数を記憶したりするなど，変数に相当するものが必要です。そこで，ムービークリップを変数として活用します。

変数用のムービークリップの作り方は簡単です。空のムービークリップを作り，1フレーム目に「Stop」アクションを入れておきます。

あとは，変数の許容範囲までフレームを伸ばすだけです。サンプルのゲームでは，ランチボックスを連続3回取り損なうと自機が減るという処理をしていますが，その3回連続で取り損なうという部分にこのムービークリップ変数が使われています。

この場合，5フレームの長さのムービークリップを作り，1フレーム目は[Stop]，2，3フレーム目は空白，4フレーム目は[Play]，5，6フレーム目は空白，7フレーム目に3回連続で取り損なったときの処理として，自機を減らすという[Tell Target]を入れました。

ランチボックスをひとつ取り損なうと，このムービークリップのフレームをひとつ進めるようにしました。そして，3回目の[Go To，次のフレーム]アクションが出たとき，4フレーム目に入り，[Play]でムービークリップが自動再生を始め，少し間をおいて，7フレーム目に突入したときに，自機を減らす[Tell Target]を実行します。これで，3回連続という回数を記憶す

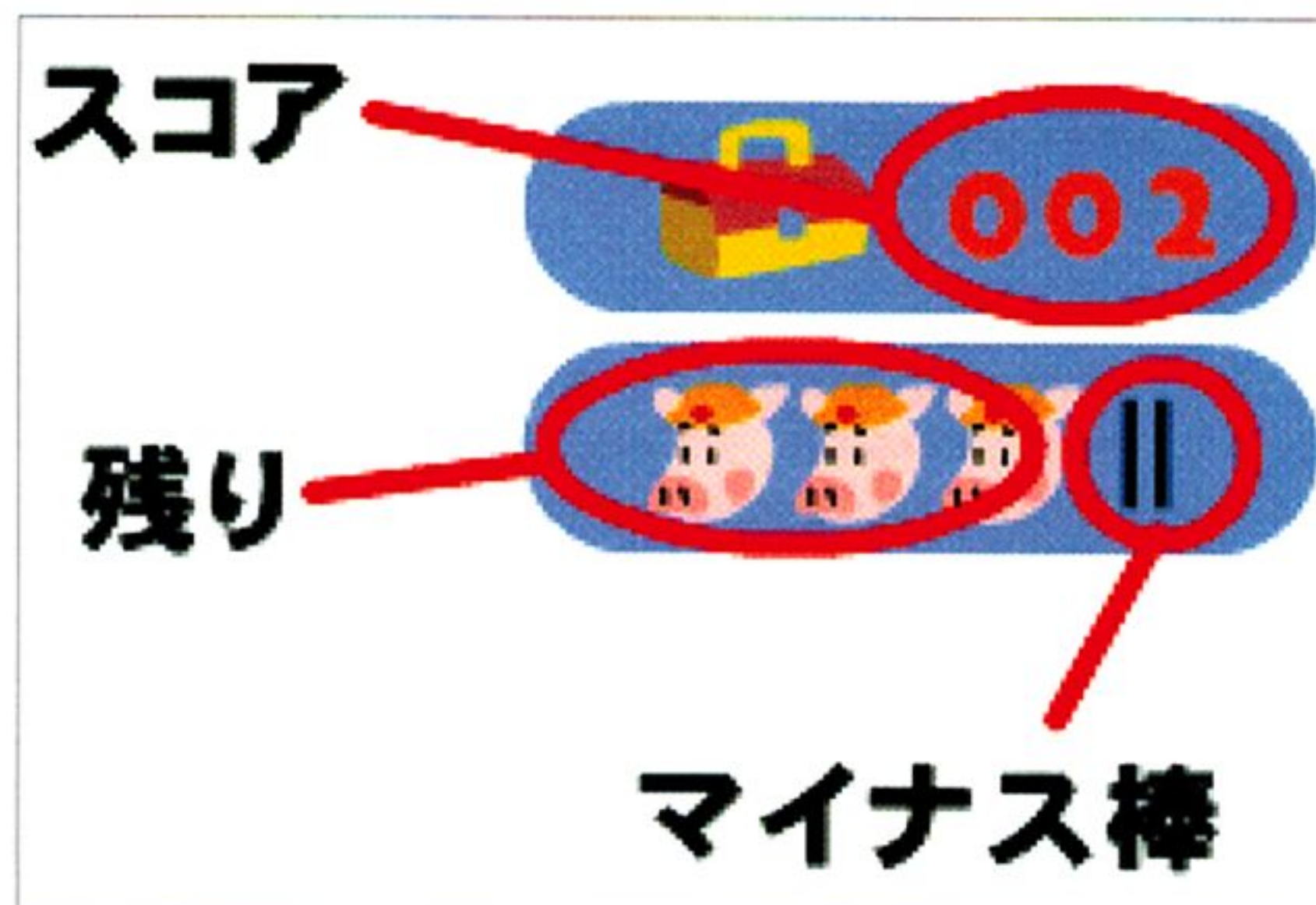



図14
3つのムービークリップ

ることができます。

ムービークリップ変数は，自機の残り数の記憶や，スコアのカウントにも使用しています。

サンプルゲーム「ヘルメッ豚98」

今回「Tell Target」を使ったサンプルゲームとして「ヘルメッ豚（とん）98」を作りました（図13）。内容は，上から落ちてくるランチボックスをひたすら取り続けるというアクションゲームです。途中，レンチが落ちてきて妨害します。うまくレンチをかわして，ランチボックスをたくさん集めましょう！　というゲームです。

では，「Tell Target」に主眼を置いて，このゲームを解析していきましょう。

Tell Targetを探す

「Tell Target」はムービークリップがあって，初めて動作することは，説明したとおりです。そして，「Tell Target」がメインに使われているこのゲームを解析するいちばんの近道は，やはり，ムービークリップを探し出すことです。そして，イベントがどこで起こるとどういう「Tell Target」が行われているのかよく観察しましょう。

もっとも簡単にそれと見分けられるのは，スコアカウントと，自機の残数表示，連続で失敗したときのマイナス棒です（図14）。

さらに，キャラクターがランチボックスを取るときの動作，レンチが当たるときの表示にも「Tell Target」を使っています。

では，それらをひとつずつ解析していきましょう。

スコアカウント

一見大変そうですが，もっとも単純な構造でできています。まず，0，1から9までの数字を1フレームごとに並べたシンボルを作ります。数字は同じ位置に配置します。これは，単純な1桁の数字のグラフィックシンボルになります。次に2桁のシンボルを作りましょう。先ほど作った1桁のシンボルを横に2つ並べた新しいシンボルを作ります。

このとき，レイヤーは別々にしてください。一の位の数字のシンボルは，インスタンスプロパティで，［ビヘイビア：グラフィック，再生モードオプション：ループ］に設定します。そして，十の位のシンボルは，［ビヘイビア：グラフィック，再生モードオプション：シングルフレーム］にします。

2つのレイヤーのフレーム数を伸ばして，十の位のシンボルのレイヤーだけ11フレーム目にキーフレームを入れます。そして，このシンボルはインスタンスプロパティで［ビヘイビア：グラフィック，再生モードオプション：シングルフレーム，最初のフレーム：2］に設定します。

［ビヘイビア：グラフィック，再生モードオプション：ループ］という設定では，数字はループ表示されますが，［再生モードオプション：シングルフレーム］では［最初のフレーム：フレーム番号］で指定されたフレームだけが表示されます。これを利用して，一の位はループでずっと表示させ，十の位は，シングルフレームで，10フレームごとに，0，1，2，3……9と増えていくようにしました。これで，100フレーム目では，（00から始まっているため）99の表示になります。

さらに，このシンボルを入れ子にして3桁を作りましょう。作り方は同じです。百の位は，1桁の数字のシンボルを［ビヘイビア：グラフィック，再生モードオプション：シングルフレーム］で表示し，十と一の位の数字は，先ほどの2桁のシンボルを［ビヘイビア：グラフィック，再生モードオプション：ループ］で配置して使います。

今回は3桁の表示まででOKですので，ここで終わりですが，4桁，5桁も同じように入れ子を繰り返せば，作成できます（図15）。通常の数字をひたすら並べる作業を考えれば，かなり効率のよい作成方法ですので，いろいろと応用が利くと思います。

スコアのシンボルは，ムービーがスタートしたときに自動再生で数字がカウントアップしていっては困りますから，アクション専用レイヤーを作って，シンボルの最初のフレームにフレームアクション「Stop」を入れます。なお，アクション専用のレイヤーを作るのは，あとで変更がしやすいためです。

図15　3桁のスコアの構造
「Tell Target」を使わなくても，FLASHのシンボルをうまく組み合わせれば作れます

自機の残数表示

自機の残数表示も，考え方はスコアカウントと同じです。1フレーム目に3匹，2フレーム目に2匹，3フレーム目に1匹のぶたを並べて，自機が減るたびに，このムービークリップを1フレーム進めればよいだけです。この作例では，4フレーム目が，ゲームオーバーの条件が揃ったときです。しかし，4フレーム目でその処理「ゲームオーバーのフレームへジャンプ」をさせずに，再生（Play）だけさせています。

これは，ぶたが0匹になった瞬間に，いきなりゲームオーバーの文字が出てくると，ユーザーがなにが起こったのか認識できないためです。「微妙な間」を稼ぐためのフレームがあるだけで，ゲームオーバーの唐突感がなくなるのです。そして，6フレーム目で本来のゲームオーバーの処理「"gameover"のラベルへジャンプ」を行っています。ムービークリップから，ステージに対して「Tell Target」を行うために，ターゲット名は，絶対パスで，ステージ「_flash0」を指しています。

ぶたの動き：ランチボックスをキャッチ

キャラクターの動きは，ほとんどをボタンで行っています。マウスカーソルを左右に動かすと，ぶたがカクカク動きながら，そのX座標を追っているように見えます。これは，「アップ」になにも入っていないボタンを作り，「オーバー」にカクカク動くぶたのムービークリップを入れ，それを画面に隙間なく並べているからです。また，取ろうとしている動作の絵を，ぶたのボタンの「ダウン」に入れています。

落ちてくるアイテムとぶたの動作が同じ（ランチボックスなら取る，レンチならぶつかるのどちらか）という点に注目して「落ちてくるアイテムのアニメーションシンボル」と「ぶたのボタンシンボル」をひとつのシンボルにしました（図16）。その結果，構造が整理され，落ちてくるアイテムのパターンやバリエーションが簡単に追加できるようになりました。

まず，アイテムが落ちてくるアニメーションをひとつのシンボルにします。次にそのシンボルと，ぶたのボタンシンボルを，もうひとつのシンボルとして作成します（lunch_downClip_OK）。これを画面に配置するだけで，アイテムが落ちてくる部分とぶたのボタンが両方できます。

さらにこの落ちてくるタイミングにあわせて，ぶたのボタンが押されたときは，ランチボックスを取る動作をさせなくてはなりません。ランチボックスを取る動作は別のシンボル「pig girl catch_OK」として用意します。

また，これは1フレーム目が空のムービークリップにしておきます。ぶた

図16　シンボルの階層化
シンボルは，何重でも階層化してひとつのオブジェクトにまとめることができます。図では3つのシンボルがひとつのシンボルに入っているのがわかります(白丸は1フレーム目が空白のシンボル)

ボタンが押されれば，そのムービークリップの2フレーム目を表示させて，ぶたがランチボックスを取っているように見せます。それと同時に，通常のボタンは消す必要がありますので，「pig girl catch_OK」の(2フレーム目の)背景を白い四角で塗りつぶして，下のボタンの絵が見えないようにしています。

このような動作をするために，このシンボルのぶたボタンの[Press/マウスダウン]アクションには以下のような「Tell Target」が書かれています。

```
On (Press)
  Begin Tell Target : c1
      Go to Next Frame
  End Tell Target
  Begin Tell Target : _flash0/score
      Go to Next Frame
  End Tell Target
  Begin Tell Target : control1
      Go to and Stop
  End Tell Target
  Begin Tell Target : _flash0/nocatch
      Go to and Stop
  End Tell Target
End On
```

注)「Begin Tell Target」の後ろにあるのは「ターゲット名」です。便宜上，このように表現しましたが，実際のアプリケーションでの表示とは異なります。

上記の中で，ひとつ目の「Tell Target」は，キャラクターの表示を切り替え，2つ目は，スコアのカウントをひとつ進めています。また，3つ目は，ランチボックスを取ったかどうかの判断のフラグ，4つ目は，「3回連続で失敗」のフラグをクリアするために，使われています。

3つ目のムービークリップ「control1」はランチボックスを取り損なったかどうかを判断する役割があります。「control1」は，このムービークリップが始まったときに初期化され，2フレーム目を指しています。

そして，後ろのフレームでは，さらに次のフレーム「3」へ進むように「Tell Target」しています。「control1」の3フレーム目には，取り損なったときの「マイナス棒」をひとつ追加するための「Tell Target」が埋め込まれています。

しかし，ランチボックスを取ると(ぶたボタンが押されると)，「control1」の1フレーム目へ戻るようにアクションが設定されていますので，3フレーム目のアクションが動くことはありません。これで，取り損なったかどうかの判断をしているわけです。

4つ目のムービークリップは，取ったときの処理として，「マイナス棒」を元に戻しています。ルールでは，3回連続で取り損なうと「マイナス棒」が3つになってぶたが1匹減ることになっていますので，一度でも取れば「マイナス棒」を初期状態(1フレーム目)へ戻す必要があるわけです。

さらに，このぶたボタンでは，[Roll Out/ロールアウト，Drag Out/ドラッグアウト]も使用しています。

```
On(Roll Out,Drag Out)
  Begin Tell Target : c1
      Go to and Stop
  End Tell Target
End On
```

ランチボックスを取った直後に，カーソルを移動させると，「c1」が表示状態のまますぐに消えず，さらに，ほかのボタンの「アップ」が出現してしまうため，ぶたが2匹同時に現れてしまいます。その対策として，ボタンからカーソルが離れるイベントで「c1」を1フレーム目に戻して，表示を消すようにしています。

ぶたの動き：レンチに当たる

ぶたがレンチに当たったときの状態は，1フレーム目が空白のムービークリップ「r1」として用意しておきます。

ぶたがレンチに当たるときの条件は，マウスカーソルがそのシンボルの上にあるすべての状態です。つまり，[Press/マウスダウン，Release/マウスアップ，Roll Out/ロールアウト，Drag Out/ドラッグアウト]の4つです。そのときのTell Targetは，

(1)ぶたがレンチに当たった絵に変更(「r1」へ[Tell Target])
(2)自機を1個減らす(「_flash0/pig_counter」へ[Tell Target])
(3)「マイナス棒」をクリア(「_flash0/nocatch」へ[Tell Target])

の3つです。また，このシンボルでも，ぶたがレンチに当たっている絵を表示しているとき(「r1」の2フレーム目を表示)に，マウスカーソルを動かすと，ぶたが2匹出現する現象が起こりますので，[Roll Out/ロールアウト，Drag Out/ドラッグアウト]によって，「r1」の1フレーム目へ戻しています。

連続で失敗したときのマイナス棒

このムービークリップ自体は，単純な構造です。段階的に，棒が1本，2本，3本と増えるようにフレームを作り，最後のフレームで，自機が減るようにフレームにアクションを組み込みます。ただし，3本目の棒が立ったときは，ユーザーにわかりやすいように，ゲームオーバーのときと同じ「微妙な間」を取るようしました。

同時出現アイテム数と「Tell Target」

このゲームでは，ランチボックスは3個，レンチは1個しか，同時に落ちてきません。それは，「Tell Target」のターゲット名は一意という制限と関係しています。たとえば，ランチボックスが落ちてくるシンボルを，タイミングをずらして2個配置したとします。ユーザーがひとつ目のランチボックスを取ろうと，マウスをクリックすると，同じターゲット名の2つ目のランチボックスのシンボル内にいるぶたまで表示されてしまいます。これは，ひとつ目と2つ目の「ランチボックスを取るぶた」のターゲット名が同じであるために起こります。

そこで，最大何個のランチボックスが同時に落ちてくるのかをあらかじめ

図17 「グリッド」と「吸着」
ゲームの完成度を上げるには正確な配置が必要です。2つのツールを上手く使いましょう。グリッド幅はムービープロパティで調整できます（メニュー「修正＞ムービー」）

図18 「ガイド」でコメントアウト
一時的にレイヤーを無効にできます。コメントアウト的に使用すると便利です

決定しておき，その数だけシンボルを複製し，ターゲット名を変えて用意しなくてはなりません。このぶたがランチボックスを取るシンボルの中では，「c1」と「control1」が該当します。サンプルでは3つの複製を作り，それぞれ「lunch_downClip_OK2」「lunch_downClip_OK3」と名づけたうえで，そのなかで使われているムービークリップのインスタンス名も，それぞれ「c2」「control2」，「c3」「control3」と変更しました（アクションのターゲット名も忘れずに変更します）。

ゲームを組み上げる

「Tell Target」を組み上げる作業とゲームの作成はほぼ同時進行で行われます。その際，もっとも注意しなくてはいけないのは，ボタンやムービークリップの配置です。少しでもずれるとその動きが気になってしまいます。

ボタンを隙間なく並べたり，正確な位置にオブジェクトを配置するときは，「グリッド」や「吸着」をうまく使いましょう（図17）。

また，不可視の状態でも，ムービープレビューでは，可視になってしまいますが，ガイドレイヤーにするとそのレイヤーを一時的に動作不可にすることができます（図18）。レイヤー単位での動作のチェックなどのテスト時には便利な機能です。もちろん，シンボルのレイアウト時には本来のガイドとしても利用しましょう。

ゲームを改造する

このゲームの基本は，なるべく簡単な構造でゲームを作成することでした。構造を簡単にすることで，ゲーム内容の変更，調整が簡単にできるからです。

アイテムの数は，レイヤーを増やすことで簡単にできます。また，アイテム出現パターンや1ステージの長さは，フレームを増やすことで対応可能となっています。

レイヤーは，「down1」「down2」「down3」が，ランチボックス専用，down4をレンチ専用に使用しています。そして，通常のぶたの表示は，「pig_btns」で行っています。

ゲームの進行は，タイムラインをそのまま利用しています。現在700フレーム程度しか使っていませんが，もっと長くすることでゲームの1ステージの時間を長くできますし，レンチのレイヤーのキーフレームを移動させて，レンチ出現のパターンを調整することも可能です。

また，落ちてくるレンチの数を増すときは，専用のレイヤーを増設して，ムービークリップも専用に増やせば簡単にできるでしょう。

ぜひ，参考にして新しいゲームを作ってみてください。

最後に

FLASHでゲームを作るといっても，要はいかに「Tell Target」を組み合わせて，複雑な処理を実現していくかということです。まだ，プログラムを組むというレベルには，ほど遠いかもしれません。

しかし，そんなFLASHで作られたゲームには，利点が3つあります。

(1) データサイズが小さく，ストリーミングができる
(2) 拡大縮小自由なアンチエイリアス表示のグラフィック
(3) (Shockwave Directorに比べて)軽いプラグインで動作

などです。

ここで，FLASHで複雑な計算や，関数が使えないなどの点が気になるという方は，JavaScriptと組み合わせて使うという手段もあります（古籏一浩さんの「JavaScriptから始めるプログラミング第2回を参考にしてください）。

FLASHでは苦手なプログラム処理をJavaScriptで行い，FLASHは表示専用のエンジンとして利用することで，さらに奥深いゲーム制作が可能になることでしょう。最後ですが，FLASHのゲーム作りに役立ちそうなURLを紹介しておきます。

FLASHのページ

・TOGORU Company
< http://www.togoru.com/ >
・MORI's Short Game Gallery
< http://www.kh.rim.or.jp/~ymori/ >

著者（伊藤のりゆき）のページ

< http://www.3oclock.com/ >

注）FLASH体験版とサンプルデータについて
Macromedia FLASH3体験版とサンプルファイルが本誌付属のCD-ROMに収録されています。興味を持った方は，ぜひインストールして実際に体験してみてください。

なお，サンプルのフォルダに収められているファイルと本記事の内容は以下のように連動しています。

Tell Target基礎

基本：sample.fla
応用1：sample+1.fla
応用2：sample+2.fla
応用3：sample+3.fla

サンプルゲーム「ヘルメッ豚98」

helmeton98.fla

*）拡張子がFLAのファイルがFLASHのソースファイルです。SWFはプラグインのインストールされているブラウザや，FLASHプレイヤーなどで再生できるShockwave Flashファイルです。

JavaScriptから始めるプログラミング　第2回
Macromedia FLASHの制御

古籏一浩　Furuhata Kazuhiro

軽快で多彩なアニメーションで動的Webページの世界に新風を吹き込んだのがMacromedia FLASHです。ここではFLASHのデータをJavaScriptから制御する方法と，その逆にFLASH側からJavaScriptを呼び出す方法の両方を見ていきましょう。

Macromedia FLASH

今回はWebでのアニメーションで多く利用されているMacromedia FLASH(FLASHムービー)を制御します。Macromedia FLASH自体については多くの書籍が出回っていますので，ここでは使い方は説明しません(プログラミングが主体ですから)。また，今号で伊藤のりゆきさんがMacromedia FLASHでゲームを作るという記事を書いていますので，そちらも参考にしてください。

さてMacromedia FLASHとJavaScriptを組み合わせるとMacromedia FLASH単体では実現が難しいことが可能となります。特にMacromedia FLASHではウィンドウの制御やレイヤーの制御(DynamicHTML制御)などを行うことができません。基本的にアニメーションに特化しているので，細かい制御はできないのです。そこでMacromedia FLASHにはLiveConnect機能が付加されています。まずは，このLiveConnect機能から説明しましょう。

図1　FLASHでアニメを作成する

LiveConnectとは？

LiveConnectは，JavaScriptがプラグインなどとのやり取りを行う機能をいいます。プラグインによってはJavaScriptからの値や命令を受け取るだけでなくJavaScriptに値を渡すことができるものもあります。残念ながらすべてのプラグインにLiveConnect機能が搭載されているわけではなく，一部のものに限られています(表1参照)。

また，LiveConnectはNetscape Navigator Communicator(以降，Netscape)ではver.3.0以上で使用できます。Internet Explorer(以下，IE)の場合はActiveX Controlになりますが，IE 3以上で使用可能です。今回制御するMacromedia FLASHはNetscape 3/IE 3以上であれば使用可能ということになります(ただし，Mac版のIE 3/4は除く)。

IE 3/4の場合，一部の命令はJavaScriptだけでなくVBScriptを必要とします。これについてはコラムに書いておきました。

Macromedia FLASHはバージョン2以上でLiveConnectが使用できます。バージョン2と3では若干機能が異なり，バージョン3ではいくつかの命令が追加されています。Macromedia FLASH 2/3で使用できるメソッドプロパティを表2に示します。

図2　アニメパターン

JavaScriptでの記述方法

JavaScriptからFLASHムービーを制御するには，あらかじめHTMLのタグにより名前をつけておく必要があります。Netscapeの場合は<EMBED>タグのNAMEオプション，IEの場合は<OBJECT>タグのNAMEオプションで名前をつけます。名前をつけておけばJavaScriptで制御することが可能になりますが，NetscapeとIEではFLASHムービーの存在するオブジェクトの階層が異なっています。そのためブラウザを判別して処理を振り分けなければなりません。

表1

```
Macromedia Flash 2/3
Macromedia Director 6～7
LiveAudio
LiveVideo
Live3D
YAMAHA MIDI PLUG (MIDPLUG for XG ver1.00はLiveConnectできない)
Javaアプレット
```

具体的には以下のようになります。FLASHムービーの名前はmyFLASHとします。

- Netscapeの場合
  document["myFLASH"]
- IEの場合
  window["myFLASH"]

NetscapeはドキュメントオブジェクトNの下にあるのに対しIEはウィンドウオブジェクトの下にあります。階層図にすると以下のようになります。

リスト1 flash.js

```
//ブラウザチェック
NS=navigator.appName.charAt (0) =="N";
MAC=!NS&&navigator.userAgent.indexOf ("Mac") >=0;
MAC=false;
if (NS) myFlashMovie=document;else myFlashMovie=window;

//LiveConnectによる制御
//Flash ver2で使用できるもの
//-----------------------------------------------------------------------------------------------------------------------
function Play () {                      if (!MAC) myFlashMovie[flashName].Play () ;                                    }
                                                               //再生
function StopPlay () {                  if (!MAC) myFlashMovie[flashName].StopPlay () ;                                }
                                                               //停止
function IsPlaying () {                 if (!MAC) alert (myFlashMovie[flashName].IsPlaying ()) ;                       }
                                                               //再生状態を返す
function GotoFrame (n) {                if (!MAC) myFlashMovie[flashName].GotoFrame (n) ;                              }
                                                               //指定したフレームへ
function CurrentFrame () {              if (!MAC) alert (myFlashMovie[flashName].CurrentFrame ()) ;                    }
                                                               //現在のフレーム数を返す
function TotalFrames () {               if (!MAC) alert (myFlashMovie[flashName].TotalFrames ()) ;                     }
                                                               //総フレーム数を返す
function Rewind () {                    if (!MAC) myFlashMovie[flashName].Rewind () ;                                  }
                                                               //巻き戻し
function SetZoomRect (x1,y1,x2,y2) {    if (!MAC) myFlashMovie[flashName].SetZoomRect (x1,y1,x2,y2) ;                  }
                                                               //指定エリアを拡大表示
function Zoom (per) {                   if (!MAC) myFlashMovie[flashName].Zoom (per) ;                                 }
                                                               //ズーム
function Pan (x,y,mode) {               if (!MAC) myFlashMovie[flashName].Pan (x,y,zoom) ;                             }
                                                               //パン
function PercentLoaded () {             if (!MAC) alert (myFlashMovie[flashName].PercentLoaded ()) ;                   }
                                                               //読み込み率
function Back () {                      if (!MAC) myFlashMovie[flashName].Back () ;                                    }
                                                               //1コマ戻す
function Forward () {                   if (!MAC) myFlashMovie[flashName].Forward () ;                                 }
                                                               //1コマ進める
function FrameLoaded (n) {              if (!MAC) alert (myFlashMovie[flashName].FrameLoaded (n)) ;                    }
                                                               //指定フレームまで読み込まれたか?
function FlashVersion (n) {             if (!MAC) alert (myFlashMovie[flashName].FlashVersion ()) ;                    }
                                                               //バージョンを返す

//Flash ver3以降で使用できるもの
function LoadMovie (Lname,URL) {        if (!MAC) myFlashMovie[flashName].LoadMovie (Lname,URL) ;                      }
                                                               //ムービー読み込み
function TGotoFrame (Tname,n) {         if (!MAC) myFlashMovie[flashName].TGotoFrame (Tname,n) ;                       }
                                                               //TellTarget指定フレームへ
function TGotoLabel (Tname,Label) {     if (!MAC) myFlashMovie[flashName].TGotoLabel (Tname,Label) ;                   }
                                                               //TellTarget指定ラベルへ
function TCurrentFrame (Tname) {        if (!MAC) myFlashMovie[flashName].TCurrentFrame (Tname) ;                      }
                                                               //TellTargetの現在のフレームを返す
function TCurrentLabel (Tname) {        if (!MAC) myFlashMovie[flashName].TCurrentLabel (Tname) ;                      }
                                                               //TellTargetの現在のラベルを返す
function TPlay (Tname) {                if (!MAC) myFlashMovie[flashName].TPlay (Tname) ;                              }
                                                               //指定したTellTargetを再生する
function TStopPlay (Tname) {            if (!MAC) myFlashMovie[flashName].TStopPlay (Tname) ;                          }
                                                               //指定したTellTargetの再生停止

//Explorer専用 (ActiveX Control)
function Stop_IE () {                   if (!MAC) myFlashMovie[flashName].Stop () ;                                    }
                                                               //停止 (Explorer)
function TotalFrames_IE () {            if (!MAC) alert (myFlashMovie[flashName].TotalFrames) ;                        }
                                                               //総フレーム数を返す
function ReadyState_IE () {             if (!MAC) alert (myFlashMovie[flashName].ReadyState) ;                         }
                                                               //読み込み状態を返す
function FrameNum_IE () {               if (!MAC) alert (myFlashMovie[flashName].FrameNum) ;                           }
                                                               //現在のムービーフレーム数を返す
function Play_IE () {                   if (!MAC) alert (myFlashMovie[flashName].Play) ;}
                                                               //現在のムービーの再生状態を返す
function Quality_IE () {                if (!MAC) alert (myFlashMovie[flashName].Quality) ;                            }
                                                               //現在のムービーの品質を返す
function ScaleMode_IE () {              if (!MAC) alert (myFlashMovie[flashName].ScaleMode) ;                          }
                                                               //拡大縮小されているかを返す
function AlignMode_IE () {              if (!MAC) alert (myFlashMovie[flashName].AlignMode) ;                          }
                                                               //行揃え状態を返す
function BackgroundColor_IE () {        if (!MAC) alert (myFlashMovie[flashName].BackgroundColor) ;                    }
                                                               //背景色を返す
function Loop_IE () {                   if (!MAC) alert (myFlashMovie[flashName].Loop) ;                               }
                                                               //現在のムービーのループ状態を返す
function Movie_IE () {                  if (!MAC) alert (myFlashMovie[flashName].Movie) ;                              }
                                                               //現在のムービーのURLを返す
-----------------------------------------------------------------------------------------------------------------------
```

・Netscapeの場合

Window
|
document
|
FLASHムービー

・IEの場合

Window
|
FLASHムービー

毎回FLASHムービーを制御するためにNetscapeとIEのプログラムを別々に用意するのは面倒です。そこで制御する前にブラウザを判別し指定

表2(メソッド/プロパティ一覧)

命令	FLASH Version	Netscape	Explorer	内容
Play()	2	○	○	アニメーションを再生する
StopPlay()	2	○	○	アニメーションの再生を停止する
Stop()	2	×	○	アニメーションの再生を停止する
IsPlaying()	2	○	○	再生中かどうかを返す
GotoFrame(フレーム番号)	2	○	○	指定フレームにジャンプする
CurrentFrame()	2	○	○	現在再生中のフレームを返す
TotalFrames()	2	○	×	総フレーム数を返す
TotalFrames	2	×	○	総フレーム数を返す
Rewind()	2	○	○	先頭フレームにジャンプする
SetZoomRect(左，上，右，下)	2	○	○	指定エリアを拡大して表示する
Zoom(パーセント)	2	○	○	指定パーセントで拡大縮小表示する
Pan(X座標,Y座標,モード)	2	○	○	ムービーをパンする
				0=ピクセル
				1=ウィンドウ
PercentLoaded()	2	○	○	ムービーの読み込まれたパーセンテージを返す。100で完全に読み込まれたことを示す
Back()	2	○	○	1フレーム戻す
Forward()	2	○	○	1フレーム進める
FrameLoaded(フレーム番号)	2	○	○	指定番号のフレームまで読み込まれたかどうか返す。読み込まれている場合はtrueを返す
FlashVersion()	2	○	○	Flashプラグインのバージョンを返す
FSCommand()	2	○	○	FlashからJavaScriptへデータを渡す
LoadMovie(レイヤー名，URL)	3	○	○	FLASHムービーを読み込む
TGotoFrame(ターゲット名，フレーム番号)	3	○	○	ターゲットのムービークリップを指定フレームに移動する
TGotoLabel(ターゲット名，ラベル名)	3	○	○	ターゲットのムービークリップを指定ラベルに移動する
TCurrentFrame(ターゲット名)	3	○	○	ターゲットのムービークリップの現在のフレーム番号を返す
TCurrentLabel(ターゲット名)	3	○	○	ターゲットのムービークリップの現在のラベル名を返す
Tplay(ターゲット名)	3	○	○	ターゲットのムービークリップを再生する
TStopPlay(ターゲット名)	3	○	○	ターゲットのムービークリップの再生を停止する
ReadyState	3	×	○	読み込み状態を返す
				0=読み込み中
				1=初期化されていない
				2=読み込み完了
				3=制御可能
				4=完了
FrameNum	3	×	○	現在のムービーフレーム。読み出し，設定可能
Play	3	×	○	現在の再生状態。読み出し，設定可能
Quality	3	×	○	ムービーの品質。読み出し，設定可能
				0=低
				1=高
				2=自動
ScaleMode	3	×	○	拡大縮小状態。読み出し，設定可能
				0=すべて表示
				1=横幅をウィンドウサイズにあわせる
				2=ウィンドウサイズにあわせる
AlignMode	3	×	○	行揃え指定。読み出し，設定可能
				Left=+1
				Right=+2
				Top=+4
				Bottom=+8
BackgroundColor	3	×	○	背景色。読み出し，設定可能
Loop	3	×	○	ループ状態。読み出し，設定可能
Movie	3	×	○	ムービーURL。読み出し，設定可能
OnProgress	3	×	○	データダウンロード率が変化したときに発生するイベント
OnReadyStateChange	3	×	○	ReadyStateが変化したときに発生するイベント

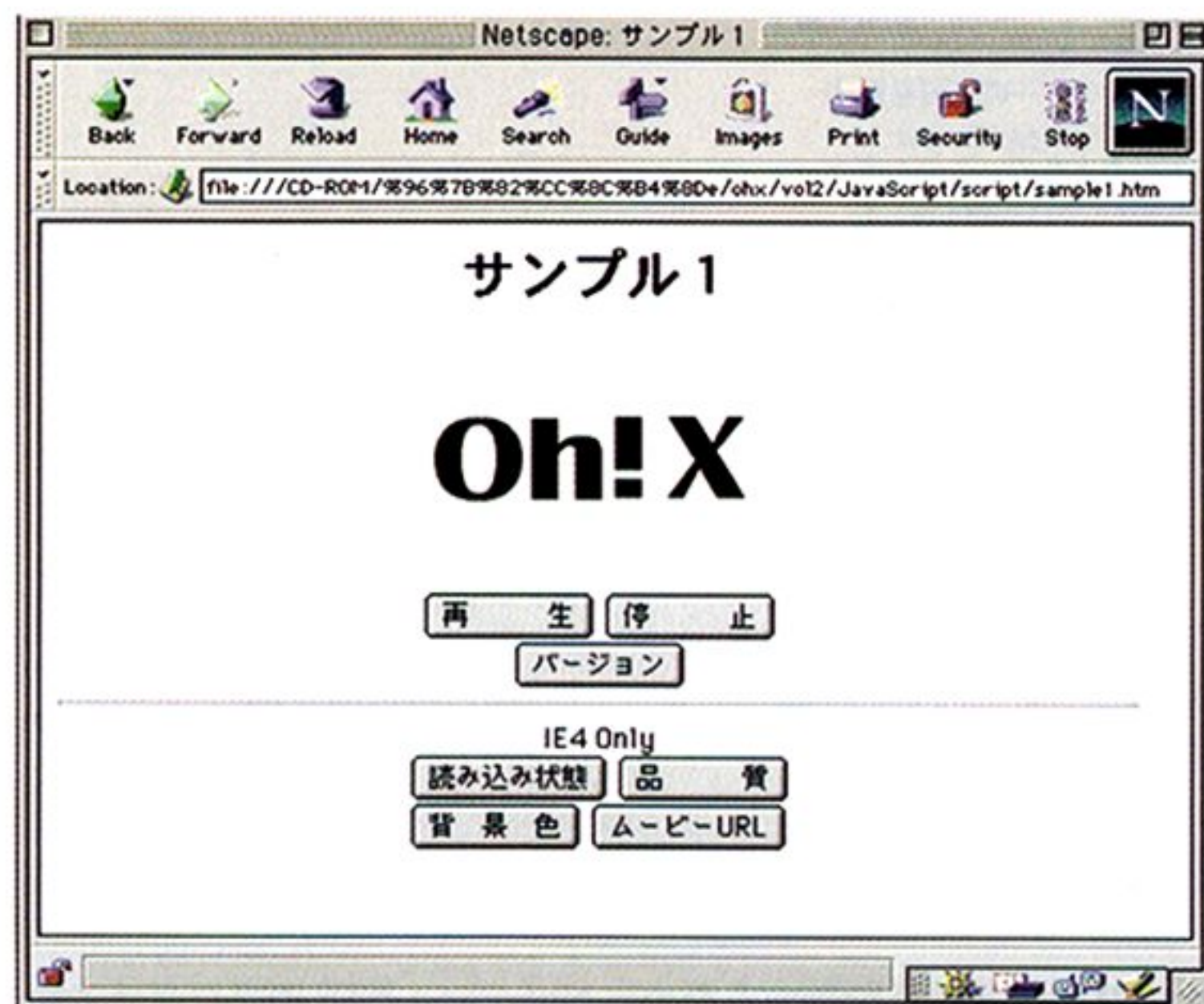



図3　実行するとこんな感じ

するオブジェクトを設定します。

まず，以下のようにしてNetscape Navigatorかどうか判別します。

```
NS = navigator.appName.charAt(0)  == "N";
```

Netscape製品であれば変数NSはtrueになります。あとはif文を使ってNetscapeならdocumentオブジェクト，IEならばwindowオブジェクトをmyFlashMovieに入れます。

```
if(NS)myFlashMovie = document; else myFlashMovie =
window;
```

このようにすれば，Netscape，IEの区別なく同じ命令でFlashムービーを制御できます。以下の例ではPlay()コマンドをFlashムービーに送っています。Flashムービーの名前はmyFLASHとしています。

```
myFlashMovie["myFLASH"].Play()
```

リスト2　Sample1.html

```
<HTML>
<HEAD>
<TITLE>サンプル1</TITLE>
<SCRIPT Language="JavaScript" SRC="flash.js"></SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  FLASHムービーの名前を設定する
    flashName = "logo";
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="#EFefFF">
<CENTER>
<H1>サンプル1</H1>
<BR>
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
CODEBASE="http://active.macromedia.com/flash/cabs/swflash.cab#version=3,0,0,0"
WIDTH="290" HEIGHT="80" NAME="logo">
<PARAM NAME="Movie" VALUE="logo.swf">
<PARAM NAME="quality" VALUE="high">
<PARAM NAME="Loop" VALUE="true">
<PARAM NAME="play" VALUE="true">
<EMBED SRC="logo.swf" NAME="logo" WIDTH="290" HEIGHT="80" LOOP="true"
QUALITY="high" SWLIVECONNECT="true">
</OBJECT>
<BR>
<BR>
<FORM NAME="myFORM">
<INPUT TYPE="button" SIZE="6" VALUE="再　　生" onClick="Play () ">
<INPUT TYPE="button" SIZE="6" VALUE="停　　止" onClick="StopPlay () "><BR>
<INPUT TYPE="button" SIZE="6" VALUE="バージョン" onClick="FlashVersion () "><BR>
<HR>
IE4 Only<BR>
<INPUT TYPE="button" SIZE="6" VALUE="読み込み状態" onClick="ReadyState_IE () ">
<INPUT TYPE="button" SIZE="6" VALUE="品　　質" onClick="Quality_IE () "><BR>
<INPUT TYPE="button" SIZE="6" VALUE="背　景　色" onClick="BackgroundColor_IE () ">
<INPUT TYPE="button" SIZE="6" VALUE="ムービーURL" onClick="Movie_IE () "><BR>
</FORM>
</CENTER>
</BODY>
</HTML>
```

リスト3　version.html

```
<HTML>
<HEAD>
<TITLE>checkPlugin</TITLE>
<script language="JavaScript">
<!--
//  FLASHプラグインバージョンを返します
//  FLASHプラグインがある場合はバージョンを返します
//  プラグインがない場合はゼロを返します
function checkFlashPlugin ( )
{
        var plugName = "application/x-shockwave-flash";// FLASHのMIME Type
        var ver = 0;
        var i;
        var pName;
        var ptr;
        var ptr2;
        if  (navigator.mimeTypes &&
             navigator.mimeTypes[plugName] &&
             navigator.mimeTypes[plugName].enabledPlugin )
        {
             for (i=0; i<navigator.plugins.length; i++)
             {
                  pName = navigator.plugins[i].description;
                  ptr   = pName.indexOf ("Flash") ;
                  ptr2  = pName.indexOf ("FlashPix") ;
                  if  ((ptr >= 0)  &&  (ptr2 < 0))  ver = eval (pName.substring (ptr+6,ptr+9)) ;
             }
        }
        return ver;
}

// -->
</script>
</HEAD>
<BODY bgColor="#ffFFff">
<CENTER>
<H2><FONT COLOR="red">Flashプラグインのバージョンを調べる</FONT></H2>
<FORM>
<INPUT TYPE="button" VALUE="プラグインチェック" onClick="alert (checkFlashPlugin ()) ">
</FORM>
</CENTER>
</BODY>
</HTML>
```

図4　ボタンを作成し配置する

図5　ポップアップメニューからFS Commandを選択する

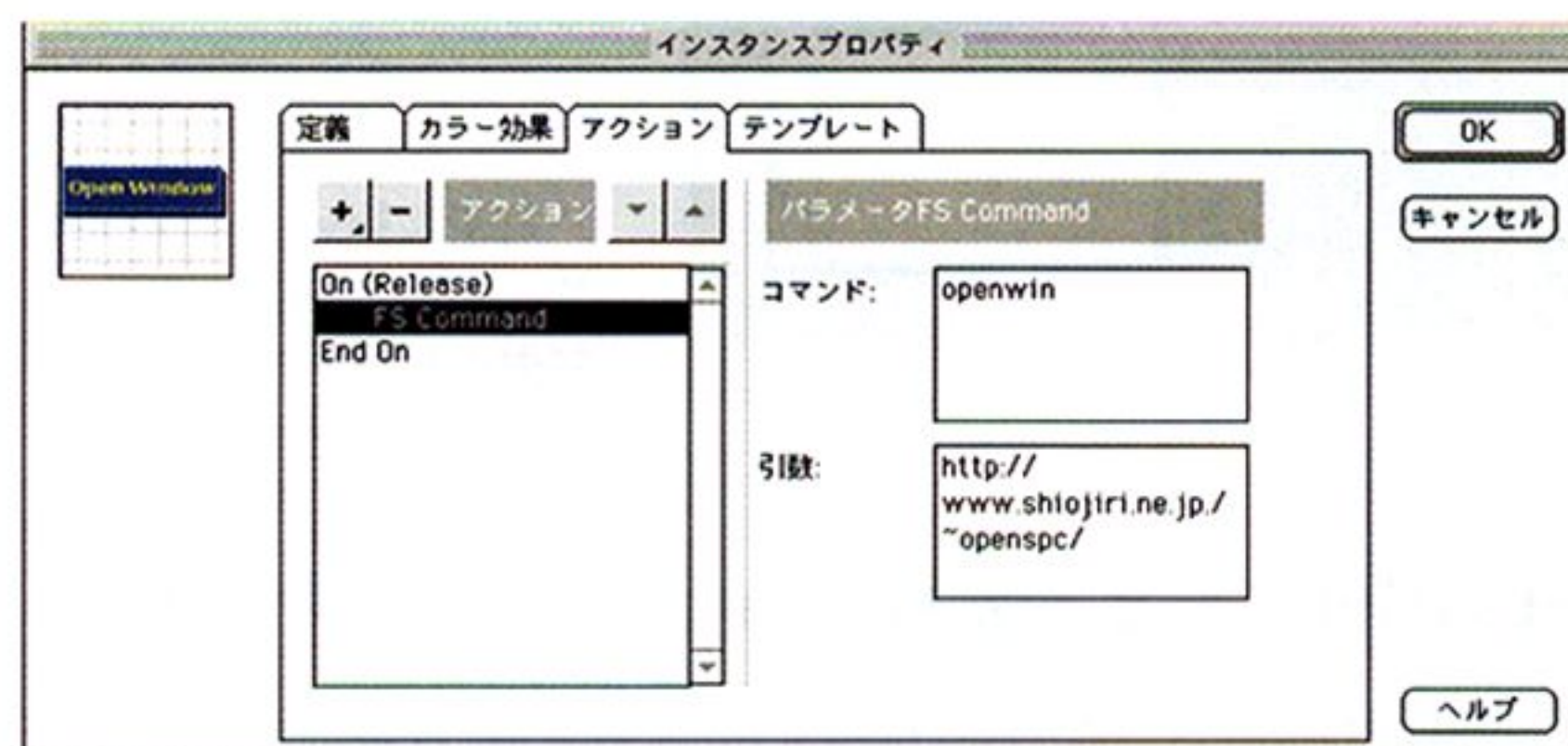



図6　コマンド，引数のところにJavaScriptに渡す値を入れる

このようにすれば制御するスクリプトも簡単に記述することができます。

さらに，もうひと工夫して制御用のJavaScriptファイルを1個にまとめておき，外部JavaScriptファイルとして呼び出すようにしておくと便利です。外部JavaScriptファイルは，

ファイル名の末尾が.jsでなければならない(拡張子.js)
純粋にJavaScriptプログラムのみ

という条件があります。今回使用する外部JavaScriptをリスト1に示します。最低限必要な設定変数はflashNameで，ここに<EMBED><OBJECT>タグでつけたFLASHムービー名を代入しておきます。

FLASHムービーを制御する

準備が整ったところでFLASHムービーを制御してみます。<OBJECT><EMBED>タグのパラメータ指定は面倒なのでMacromedia FLASHに付属してくるAftershockを使って作成するのがよいでしょう。注意点としては<OBJECT><EMBED>タグのNAMEオプションでつける名前です。この名前がLiveConnectで制御する場合の名前になります。

あとは単純にFLASHムービーを制御するメソッドを呼び出すだけです。表2にあるようにFLASHプラグインのバージョン，ブラウザにより使用できるメソッド/プロパティが異なりますので使う前に確認しましょう。多く

リスト4　Sample2.html

```
<HTML>
<HEAD>
<TITLE>サンプル2</TITLE>
<SCRIPT Language="JavaScript" SRC="flash.js"></SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  ここがFSCommandを処理する部分
function fsc_DoFSCommand (cmd,theURL)
{
        newWin = window.open (theURL,"Swin","width=320,height=240") ;
}
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="#EFefFF">
<CENTER>
<H1>サンプル2</H1>
<BR>
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
CODEBASE="http://active.macromedia.com/flash/cabs/swflash.cab#version=3,0,0,0"
WIDTH="290" HEIGHT="80" NAME="fsc">
<PARAM NAME="Movie" VALUE="fscmd.swf">
<PARAM NAME="quality" VALUE="high">
<PARAM NAME="Loop" VALUE="true">
<PARAM NAME="play" VALUE="true">
<EMBED SRC="fscmd.swf" NAME="fsc" WIDTH="290" HEIGHT="80"
LOOP="true" QUALITY="high" SWLIVECONNECT="true">
</OBJECT>
</CENTER>
</BODY>
</HTML>
```

図7　実行するとこんな感じ

図8　ムービークリップを作成し配置する

の人に見てもらいたいのであれば共通のメソッド/プロパティを使うべきでしょう。どうしてもバージョンをチェックして処理したい場合はリスト3のようなバージョンチェックプログラムを使ってください。

実際にLiveConnectを使ってFLASHムービーを制御するプログラムをリスト2に示します。

FLASHムービーからJavaScriptを呼び出す

今度はFLASHムービーからJavaScriptを呼び出してみます(Explorerの場合はVBScriptも必要。コラム参照)。

FLASHムービーからJavaScriptを呼び出すにはFS Commandを使います。まず，図4のようにボタンを作成し配置します。ボタン上でダブルクリックすると図5のようなウィンドウが表示されます。アクションタブをクリックし「+」ボタンを押します。表示されたメニューの中からFS Commandを選択します。選択すると右側にパラメータFS Command欄が表示されます。「コマンド」「引数」の入力欄がありますので，ここにJavaScriptに渡したい値を記述します。コマンド，引数と分かれていますが，便宜上分類されているだけでコマンドの部分に引数を入れても問題ありません。

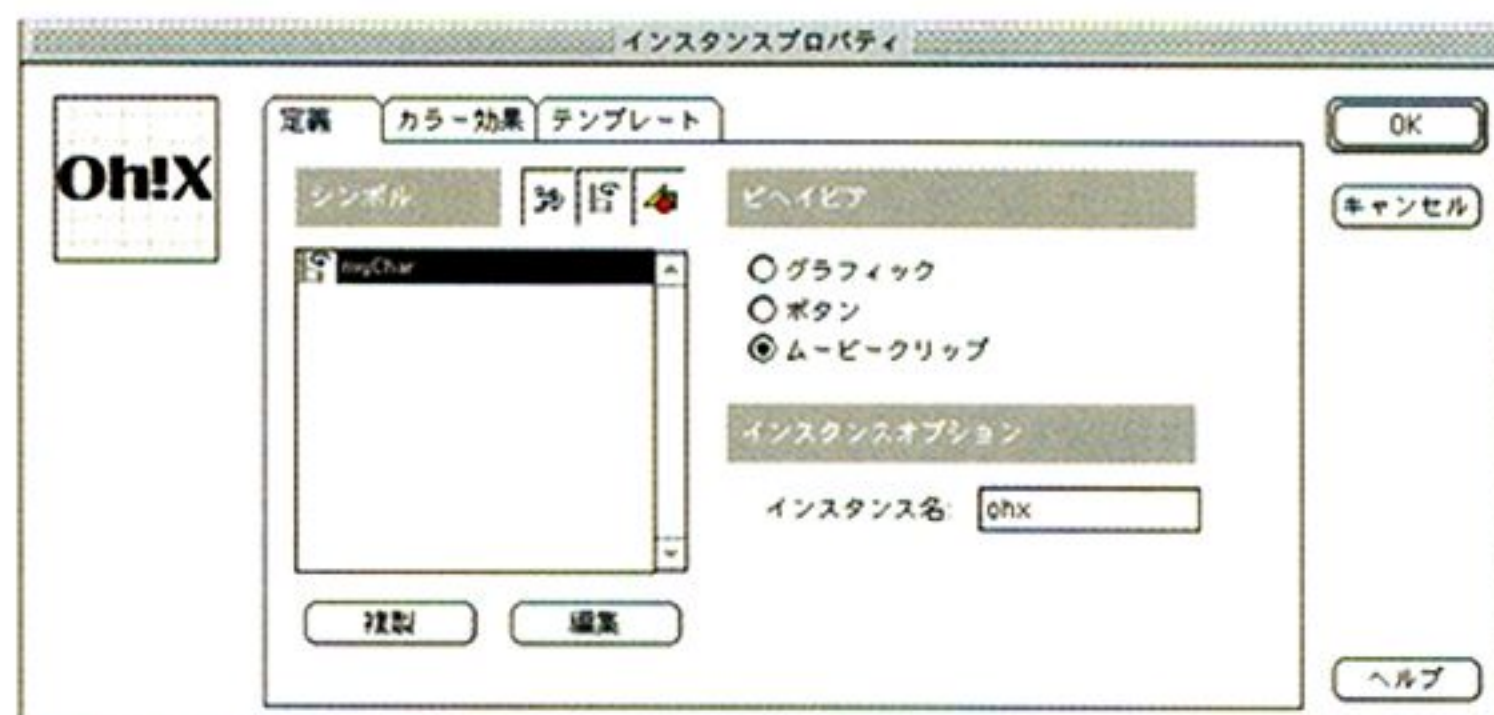



図9　ダブルクリックし定義タブを押し，インスタンス名を入力する

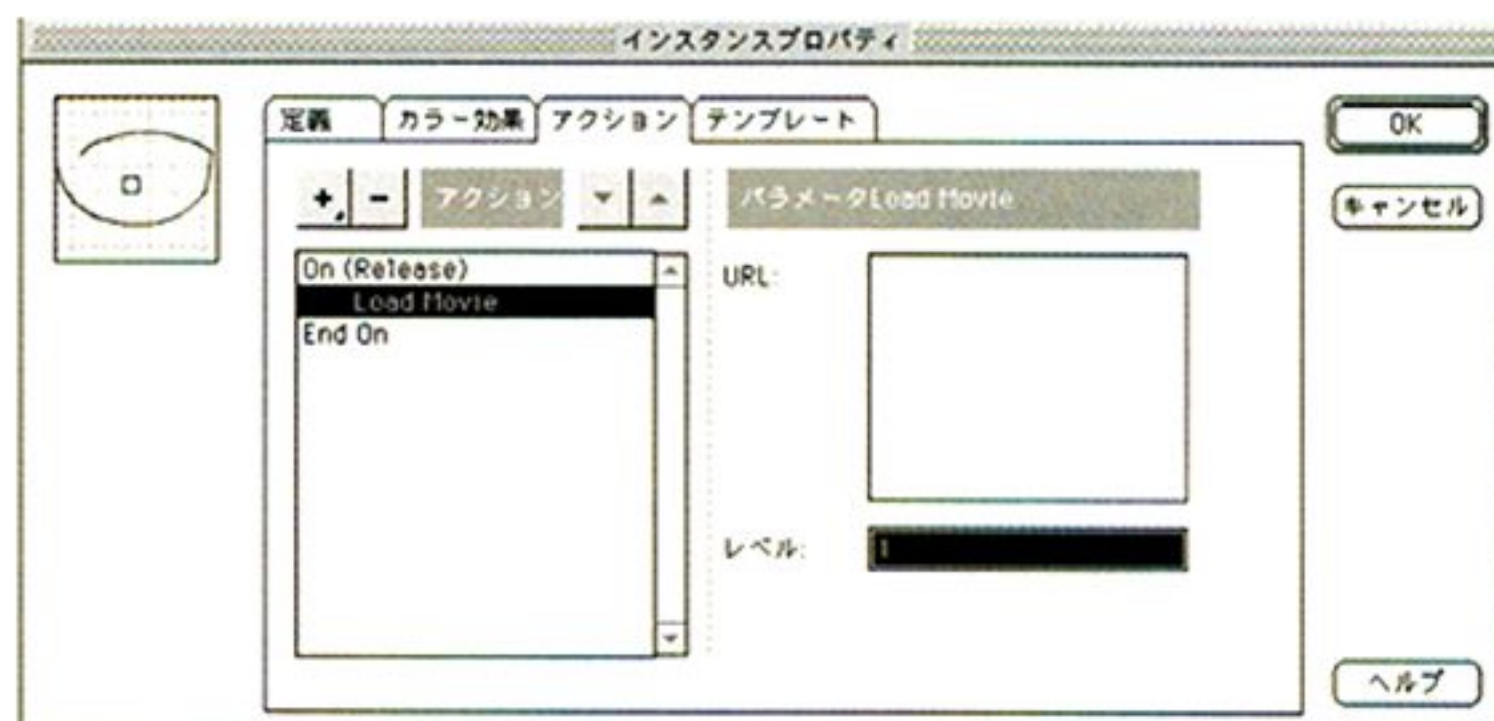



図10　Load Movieの指定

これでFLASHムービー側の設定は終わりです。FLASHムービー上でボタンが押されるとJavaScript側ではDoFSCommand関数が呼び出されます。しかし，DoFSCommandという関数を用意しただけでは駄目です。ここには，ちょっとした仕掛けがしてあり，DoFSCommand関数名の前に<EMBED><OBJECT>タグのNAMEオプションで指定されたFLASHムービー名を_(アンダーバー)で区切ってつけるのです。たとえば，myMovieという名前のFLASHムービーであれば，

```
myMovie_DoFSCommand(){ …… }
```

となります。複数のFLASHムービーからDoFSCommandを呼び出す場合，ひとつのDoFSCommand関数内で振り分けるのではなく，最初から振り分けるようにしてあるわけです。

今度はFLASHムービーから受け渡される引数です。これは，

```
myMovie_DoFSCommand(command,argument){ … }
```

といった「コマンド(command)」「引数(argument)」の2つがJavaScriptに渡されます。commandがFLASHムービー作成時に指定した「コマンド」で，argumentが「引数」にあたります。

リスト4ではFSCommandを使って新しくウィンドウを開き，指定されたURLのページを読み込むものです。

注意しなければいけないのは<EMBED>タグのSWLIVECONNECTオプションです。これはFSCommandを使用する場合に指定しておく必要があります。Netscapeの場合LiveConnect機能を実現するためにJavaを使用しています。そのためJavaが起動していないとLiveConnectが機能しません。このSWLIVECONNECTオプションはJavaが起動していない場合，強制的にJavaを起動させてくれます。FSCommandを使用しない場合は特に指定する必要はありません。

FLASH3で追加された機能を使う

最後にMacromedia FLASH 3で追加されたTell targetを使ってみます。JavaScriptからTell targetを使いムービークリップを制御することにします。まず，新しいシンボル(ムービークリップ)を作成します。ここではOh!Xのロゴが左回転するようにしました。作成したシンボルを図8のようにムービーに配置します。配置したシンボルをダブルクリックします。すると図9のようなインスタンスプロパティウィンドウが表示されます。図9の

図11　実行するとこんな感じ

ような画面にならないときは定義タブをクリックしてください。インスタンスオプションのインスタンス名に名前を入力します。ここで入力した名前がJavaScript側で制御するときの名前になります。

これでFLASHムービー側の準備は終わりです。次はJavaScript側ですが，先ほど作成したFLASHムービーを制御するのと同じです。ただ注意しなければいけないのはインスタンス名です。単純にインスタンス名をTell Targetで制御する名前として渡しても動作しません。Load Movieによって読み込まれたムービーレベルに応じたパス名をつける必要があるためです（図10参照）。

今回のようにLoad Movieを使わない状態であれば，

_flash0/ターゲット名

として指定します。ターゲット名がohxであれば，

_flash0/ohx

となります。ここでLoad Movieを使ってムービーがレベル1に読み込まれた場合は，

_flash1/ターゲット名

のようになります。

リスト5　Sample3.html

```
<HTML>
<HEAD>
<TITLE>サンプル3 </TITLE>
<SCRIPT Language="JavaScript" SRC="flash.js"></SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  FLASHムービーの名前を設定する
        flashName = "logo";
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="#EFefFF">
<CENTER>
<H1>サンプル3 </H1>
<BR>
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
CODEBASE="http://active.macromedia.com/flash/cabs/swflash.cab#version=3,0,0,0"
WIDTH="290" HEIGHT="80" NAME="logo">
<PARAM NAME="Movie" VALUE="ttarget.swf">
<PARAM NAME="quality" VALUE="high">
<PARAM NAME="Loop" VALUE="true">
<PARAM NAME="play" VALUE="true">
<EMBED SRC="ttarget.swf" NAME="logo" WIDTH="290" HEIGHT="80" LOOP="true"
QUALITY="high" SWLIVECONNECT="true">
</OBJECT>
<BR>
<BR>
<FORM NAME="myFORM">
<INPUT TYPE="button" SIZE="6" VALUE="再生" onClick="TPlay ('_flash0/ohx') ">
<INPUT TYPE="button" SIZE="6" VALUE="停止" onClick="TStopPlay ('_flash0/ohx')
"><BR>
</FORM>
</CENTER>
</BODY>
</HTML>
```

終わりに

Macromedia FLASHとのLiveConnectを取り上げてみましたが，もっとも細かいことができるのがJavaとのLiveConnectです。次はMacromedia Directorでしょう。Macromedia FLASHではJavaScriptからシンボルの座標が指定できませんが，Macromedia Directorであれば問題なくできます。Macromedia FLASH 4にJavaScriptから座標を指定できるとか，透明度を指定できるなどのLiveConnectコマンドが追加されれば面白いかな，と思いますが実際にどうなるかはわかりません。

LiveConnectなどについても私のWebページで若干用意してありますので参考にしてください。

http://www.shiojiri.ne.jp/˜openspc/JavaScript

COLUMN　IEでFSCommandを受け取る

Internet Explorer 3/4でうまく動作しない命令にFSCommandがあります。FLASHからJavaScriptに値を渡す命令ですが，IEではFSCommandを受け取ることができません。そこでVBScriptで値を受け取ってからJava Scriptを呼び出すという方法を使います。これはマクロメディアのWebページにも記述されています。

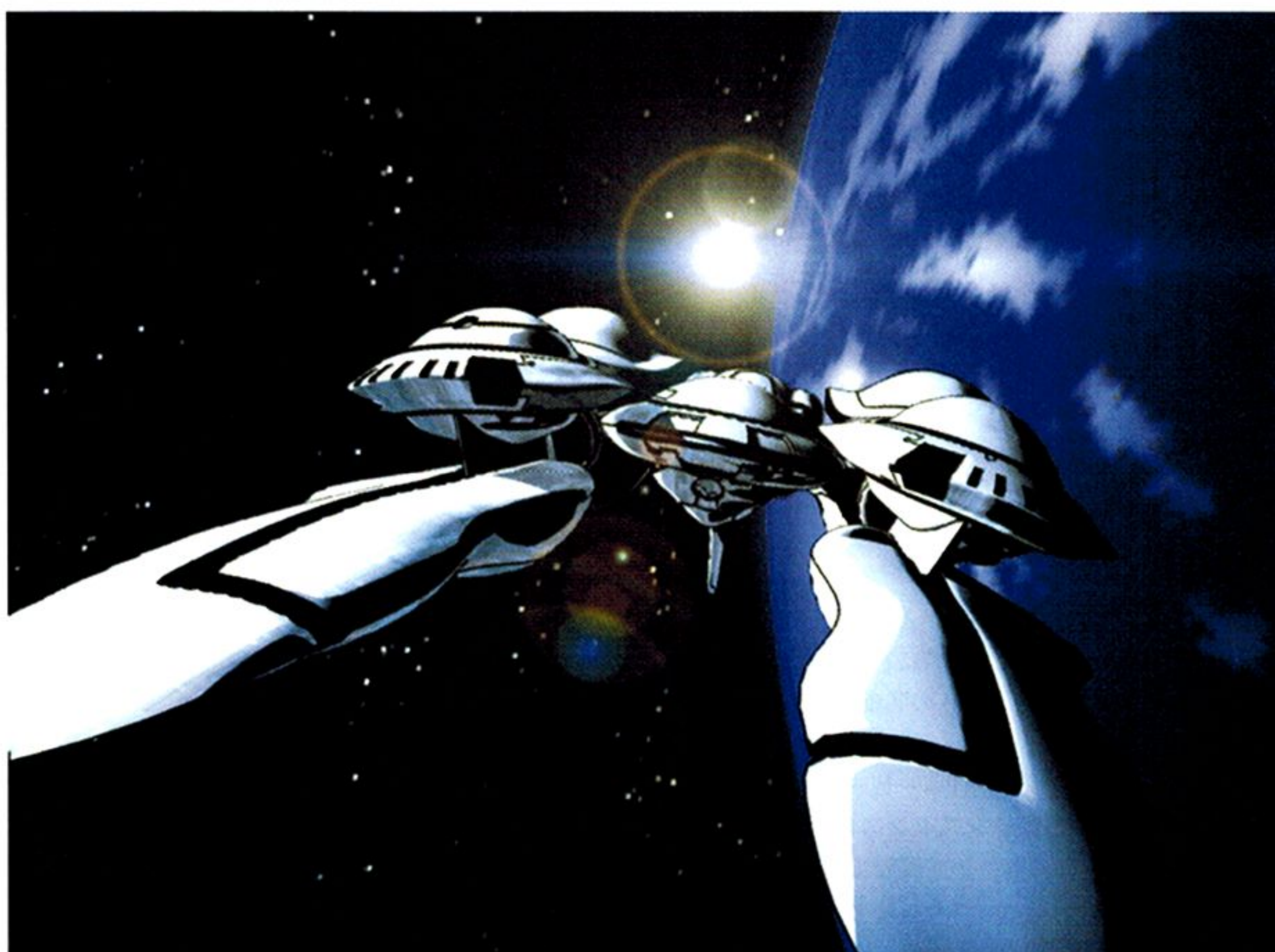
リスト4をIEでも動作するように書き換えるには，以下の命令を付け加えます。リスト6が実際のスクリプトになります。

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub fsc_FSCommand (ByVal cmd, ByVal theURL)
        call fsc_DoFSCommand (cmd,theURL)
end sub
-->
</SCRIPT>
```

リスト6　Sample4.html

```
<HTML>
<HEAD>
<TITLE>サンプル2 </TITLE>
<SCRIPT Language="JavaScript" SRC="flash.js"></SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  ここがFSCommandを処理する部分
function fsc_DoFSCommand (cmd,theURL)
{
        newWin = window.open (theURL,"Swin","width=320,height=240") ;
}
// -->
</SCRIPT>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub fsc_FSCommand (ByVal cmd, ByVal theURL)
        call fsc_DoFSCommand (cmd,theURL)
end sub
-->
</SCRIPT>
</HEAD>
<BODY bgColor="#EFefFF">
<CENTER>
<H1>サンプル2 </H1>
<BR>
<OBJECT CLASSID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
CODEBASE="http://active.macromedia.com/flash/cabs/swflash.cab#version=3,0,
0,0"
WIDTH="290" HEIGHT="80" NAME="fsc">
<PARAM NAME="Movie" VALUE="fscmd.swf">
<PARAM NAME="quality" VALUE="high">
<PARAM NAME="Loop" VALUE="true">
<PARAM NAME="play" VALUE="true">
<EMBED SRC="fscmd.swf" NAME="fsc" WIDTH="290" HEIGHT="80" LOOP="true"
QUALITY="high" SWLIVECONNECT="true">
</OBJECT>
</CENTER>
</BODY>
</HTML>
```

project team DoGA

かまたゆたか

テレビアニメ(CGパート)の作り方

皆さんは，テレビアニメの制作に参加されたことがあるでしょうか？……
普通ないわなぁ。そんなこと，生涯あるわけないわなぁ……と，私も思っていました。
そんな私たちが，突然テレビアニメのCGを作ることになったのです。
そう，昨年4月から9月までテレビ東京系列で放映された「ロストユニバース」です。
受注から放映まで，素人の目で見たテレビアニメ制作の舞台裏を紹介しましょう。
[この記事に関するホームページ http://www.doga.co.jp/codoga/lostuniv/]

いかにして受注するか

DoGAはCGアニメ文化の振興や啓蒙を目的に活動しているアマチュアの団体です。それが6年前に，税金や法律の都合で，子会社を設立しました。これが株式会社ドーガです。なにしろ親団体がアマチュアですから，会社といってもあまり積極的に営利活動はしていません。しかしCGに関する技術やCG映像制作能力はあるので，ゲームのデータやオープニングアニメの制作はよく受けていました。

しかし，テレビアニメとはまったく縁などありません。そんな会社がどうやって，テレビアニメ制作の仕事を受注したかというところからお話ししましょう。

日本でテレビアニメの仕事を受注する場合，まず「全日本テレビアニメ制作協会」に加盟しなければいけません。そのためには，会社の規模や経営状態，過去の実績などの面で厳しく審査され，なおかつ協会に加盟している会社2つ以上から推薦をもらわないといけません。

加盟すると，年6回，奇数月に協会が主催する通称"セリ市"に参加することができます。この"セリ市"では，各テレビ局から企画書が発表され，放映開始予定日や内容などの細かい資料が提示されます。それに対して制作会社が，受注したい番組に希望の値段をつけ，セリ落としていきます。

……などという話はウソです(オイオイ)。結論からいうと，テレビアニメ業界は，この例のような秩序なんかとは無縁の世界です。いい加減とデタラメが支配する世界なのです。

あれは，去年のお正月休みのことです。CG関係の仕事をされている知人のAさんがDoGAに遊びにきました。

かまた：どないでっか。もうかってまっか

A：ぼちぼちでんなぁ。今年は，テレビアニメを制作するかもしれませんわ
かまた：へぇ，面白そうやん。まさかフルCG？
A：とんでもない。毎週数カットだけです
かまた：それでも，毎週毎週作り続けるのなんて，考えただけでも大変そうやねぇ。まぁ，体壊さん程度にがんばらな(完全に他人事モード)
A：でも，ほかにも仕事があって，大変なんすよ
かまた：エライたくさん仕事があって，うらやましいねぇ。今度なんか仕事回してよ

……などと世間話をしておりました。そして，1月末，そのAさんから電話が。
A：かまたさん，例の件，DoGAでやってよ
かまた：はて，なんの話やら?(すでに忘れている)

……これで受注が決定しました。

打ち合わせもひと苦労

いまでこそテレビアニメの中でCGが用いられるのも珍しくありませんが，その当時はまだ少なく，今回のように毎週新作のCGシーンがあるというのは前例のないことでした。特に，大手のアニメ制作会社の場合，社内にCGの部署を設けて制作するため，連携などがしやすいのですが，今回のように毎週のCGパートを外注するというのは，テレビアニメ史上初の試みなんだそうです。

ですから，我々も制作会社もわからないことだらけでした。当方はテレビアニメ制作についてはまったくの素人ですし，アニメ制作会社もCGに関する知識はほとんどありません。こうなると，打ち合わせをしてもさっぱり話が通じないのです。
DoGA：御社ではCGは初めてですか？
制作会社：いえ，去年から導入しています
D：あっそうなんですか。どんなソフトを使用しておられるのですか？
制：「アニモ」です。デジタル彩色をやってます
D：???

解説：デジタル彩色とは，セルに絵の具で色を塗る代わりに，スキャナで取り込み，アニモというアニメ専用のペイントソフトで，コンピュータ上で色を塗る作業。制作会社側は，コンピュータを使う作業のすべてをCGと認識しているようだ

D：モーションデザインに入る前に，モデリングの作業の期間が必要です
制：モデリングとモーションデザインとは別の作業なんですか？
D：別です。モデリングで物体を作り，そのデータをカットごとにモーションデザインします
制：モデリングのデータは何度も使うのですか？
D：もちろん使います
制：ではそれはバンクなんですね
D：???

解説：バンクとは，たとえば魔女っ子モノの変身シーンなど，毎回のように何度も流用するようなカット。もうこうなると，制作会社がモデリングやモーションデザインをどう勘違いしているのか，想像することもできない

お互いの専門用語がよくわからない。お互いどんな作業が必要なのかよくわからない。そのなかで，制作の一部を分担しようというのです。打ち合わせをするほどに，こりゃなかなかヤッカイだなぁという雰囲気が漂ってきました。

ただ，双方やる気はありました。やってみないとわからない。どんなことが，どの程度の量できるのかは責任が持てない。でも，とにかくやってみようじゃないかということで，前向きに検討していきました。

忘れちゃいけないお金の問題

テレビアニメ業界全体が異常に低賃金だということはご存じでしょうか？　その昔，故手塚治虫先生が，日本で初めてテレビアニメを制作されたとき，その制作費をまともに請求すると，高すぎて，どこのテレビ局も買ってくれませんでした。手塚先生としては，なんとか放映したいという想いが先に立ち，無茶な安値に設定したところ，その価格がその後のテレビアニメの基準になってしまったのです。

たとえば，知人のBさんは，大阪の某大手CGプロダクションで働いていたところ，アニメ制作会社から声がかかりました。特別待遇をするから，ぜひ東京にきてくれというのです。テレビアニメに興味を持っていたBさんは，さっそく見学に行ったところ，月給は10万円といわれました。Bさんが「東京で10万円でどうやって暮らしていくんだ」と怒ったら「CGの腕を見込んでの特別待遇だ。普通なら7万だぞ」といわれたとか。

こういった話を聞いていたので，今回のCGパート制作費はかなり安いだろうと覚悟はしていました。それでも，ちょっとびっくりするような値段が提示されました。

皆さんは，テレビアニメの制作費(30分1話分)って予想できますか？　もちろん，ピンからキリまであります。でも，〝放映時間帯やテレビ局によってだいぶ差があるだろう〟と思うのは間違いです。たとえば，この「ロストユニバース」は，テレビ東京系列の金曜の6：30からでしたが，同じ局でその直前の6：00から放映していた「カウボーイ・ビバップ」の制作費は8倍強というウワサです。つまり，スポンサーの力，そして，ゲーム化，ロボットなどのキャラクターの商品化，劇場化などの有無で何倍も違ってくるのです。その点，「ロストユニバース」にはなんの商品化の話もなく，完全にキリのほうでした。

やりたいのはやまやまですが，だからといってあんまり安い予算で引き受けると，手塚先生のようにCG業界の悪い前例になってしまいます。そこで，制作会社と一緒になって，CGパートに最大いくらの予算が取れるか計算し直しました。CGパートは全体の何パーセントか，いらなくなるセルの枚数，不要になる作業，人件費……。

それでも，たいした金額にはなりません。制作期間中，うちのスタッフ2名を専任させると赤字になってしまいます。しかたがないので，専任制作スタッフはたったのひとり，そして期間前半だけモデリング用スタッフを追加するという実に弱小の体制で臨むことになりました。

さらに，DoGAではその金額内でできることしかやらない，また今回の金額は今後のCG制作の参考にしないという条件をつけました。

いよいよ制作開始

CG制作スタッフは，モーションデザインが宇宙人森山さん(代表作「Epa2ビデオマニュアル」，6thオープニング「ダイダロス」)で，モデリングのエキスパートとして渡辺哲也さん(9th「ゼノヴァー」，10th「リューセイバー」)にもご参加いただきました。コストパフォーマンスでは最強のコンビといえるでしょう。本当は，エフェクト担当として青山さん(9th「WIVERN」)も呼びたかったのですが，予算が足りず，スケジュールもあわなかったので断念しました。

渡辺さんに参加してもらったのにはもうひとつ理由があります。今回の依頼の条件のひとつとして，セル画との違和感をなくすために，セルっぽいCGにしないといけません。その点，LightWave 3Dのセルシェーダーの先駆者である渡辺さんのご指導をいただければ完璧だと考えたわけです。

さて，スタッフを集める間にも，制作に関する細かな内容や条件などを決め，契約書を交わそうとしました。しかし，なんとアニメ業界では，制作依頼をするのにいちいち契約書など交わさないというのです。将来の無用なトラブルを避けるため，契約書は絶対必要です。そこで，「悪い慣習ってのは，受け継ぐ必要なんてないんですよ」と先方の社長を説得し，快く了承していただけました。

こうして2月上旬，DoGA内でテレビアニメのCG制作がスタートしました。しかし，このとき既に放映開始まで2カ月を切っています！

D：まず，どんなシーンから作らないといけないのでしょう？
制：最初の数話は，主人公の宇宙船ソードブレイカーしか出てきません
D：わかりました。ではすぐにソードブレイカーをモデリングします。至急デザインなどの資料を送ってください
制：いえ，デザインはまだ決定していません

本当に大丈夫なのか？

第1カットはワープシーン

渡辺さんの驚異的な尽力により，宇宙船ソードブレイカーのモデリングは，予定よりずっと早く完了しました。早速モーションデザインに移りたいところですが，今度は第1話の絵コンテがきません。

制：コンテはまだしばらくかかります。とりあえずバンクで使用するワープシーンを作ってください
D：わかりました。では，どんなワープシーンにしましょう
制：そうですね。資料を送ります

数日後，資料が送られてきました（画像1）。これだけ……。

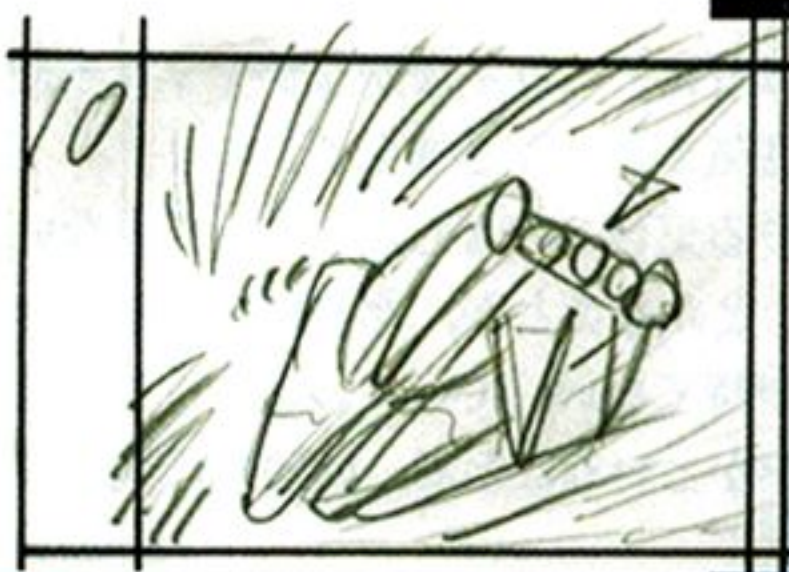
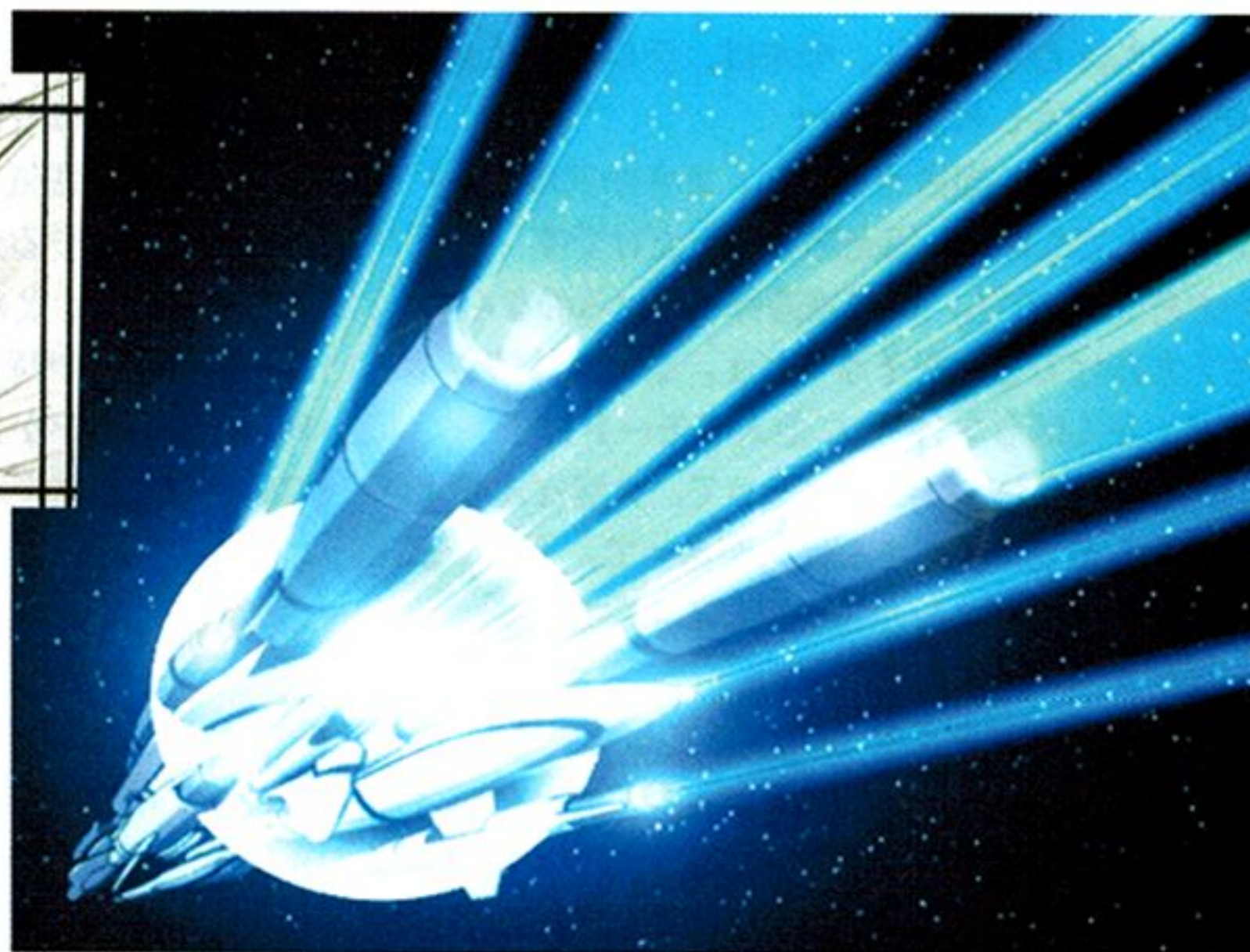
注：この絵はイメージを再現したものです。各種資料は制作会社の著作物であり，契約上この場では公開できません

こんなんじゃ，ちっともわからん！　しかたがないので直接監督に伺ってみました。

監督：スタートレックみたいな感じです。グイーンと伸びて，ズバン！

やっぱりこれだけ……。ということで，こちらで適当に作ってしまいました（画像2）。

つまり，アニメ制作では，あれこれ細かい指定なんてものはないんです。かなり自由な裁量を許されているといえます。でもこれは，両者の信頼の上に成立している……なんてカッコいいものではなく，単に細かい指示をしている余裕がないんでしょうねぇ。

実際，本格的に制作に入っても，与えられる指示や情報は乏しく，正確にわかっているのは秒数だけ。あとは絵コンテの非常にラフな絵から，意図されている動きを推測することを要求されます。

画像1　ワープシーン作成用資料はこれだけ。これだけでいったいなにがわかるんだぁ！

画像2　完成したワープシーン。一応スタートレック風ということでこんな感じ

オープニング秘話

CGの準備期間も短かったのですが，セルのほうも大変だったようです。特にオープニングアニメの制作は大幅に遅れました。そして放映開始の時点で完成しませんでした。ではどうなったかというと，一部「ただ今制作中です」と表示され，主人公が頭を下げている絵が流れたのです。たぶん，あまり例がないのではないでしょうか。

そんな状況なので，当然DoGAにもオープニングを手伝ってくれという要請がきます。

監督：オープニング用のCGカットをいくつか作ってください

かまた：しかし，オープニングの制作は契約にありません

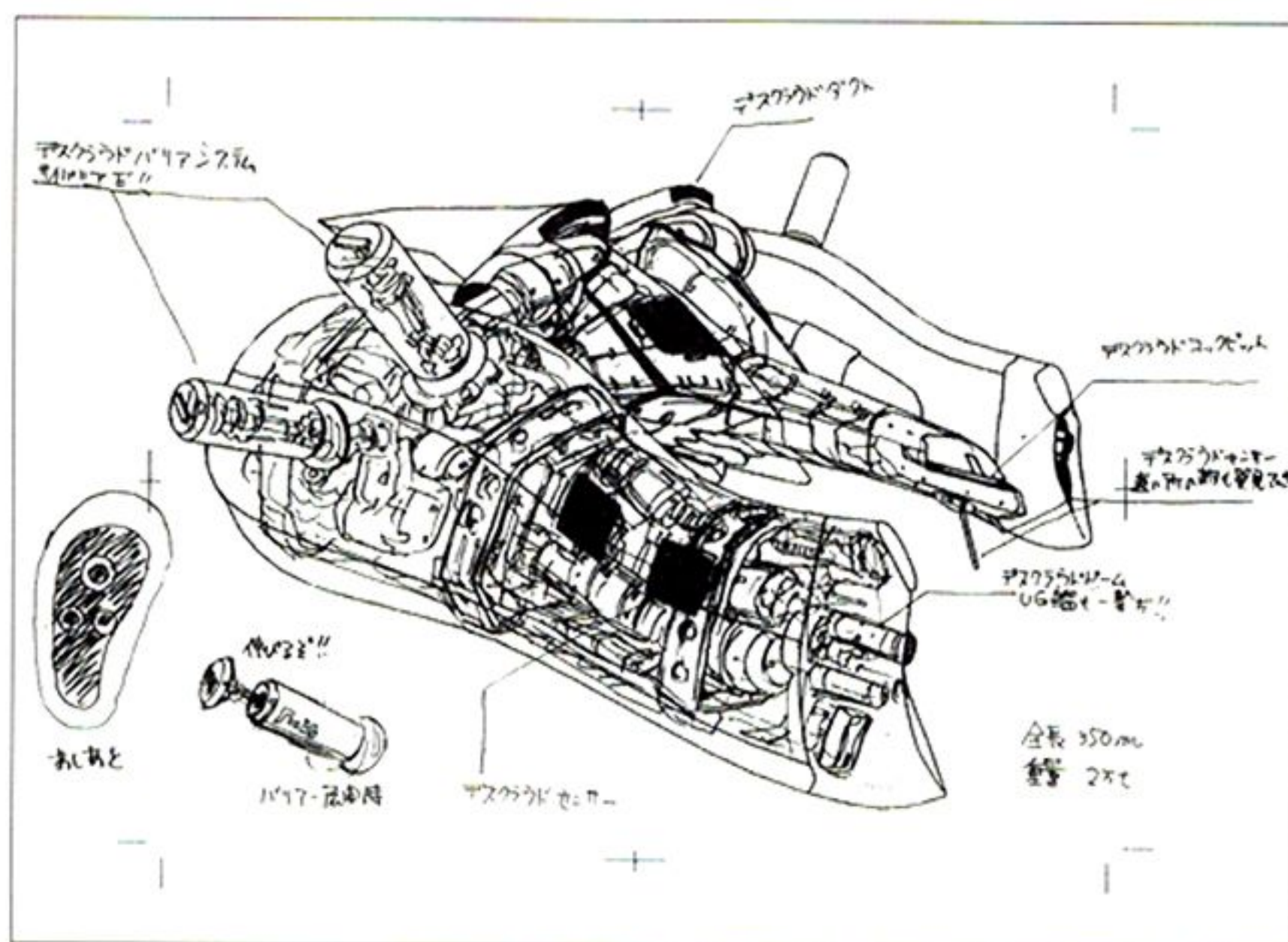
監：そうですよね。無理ですよね

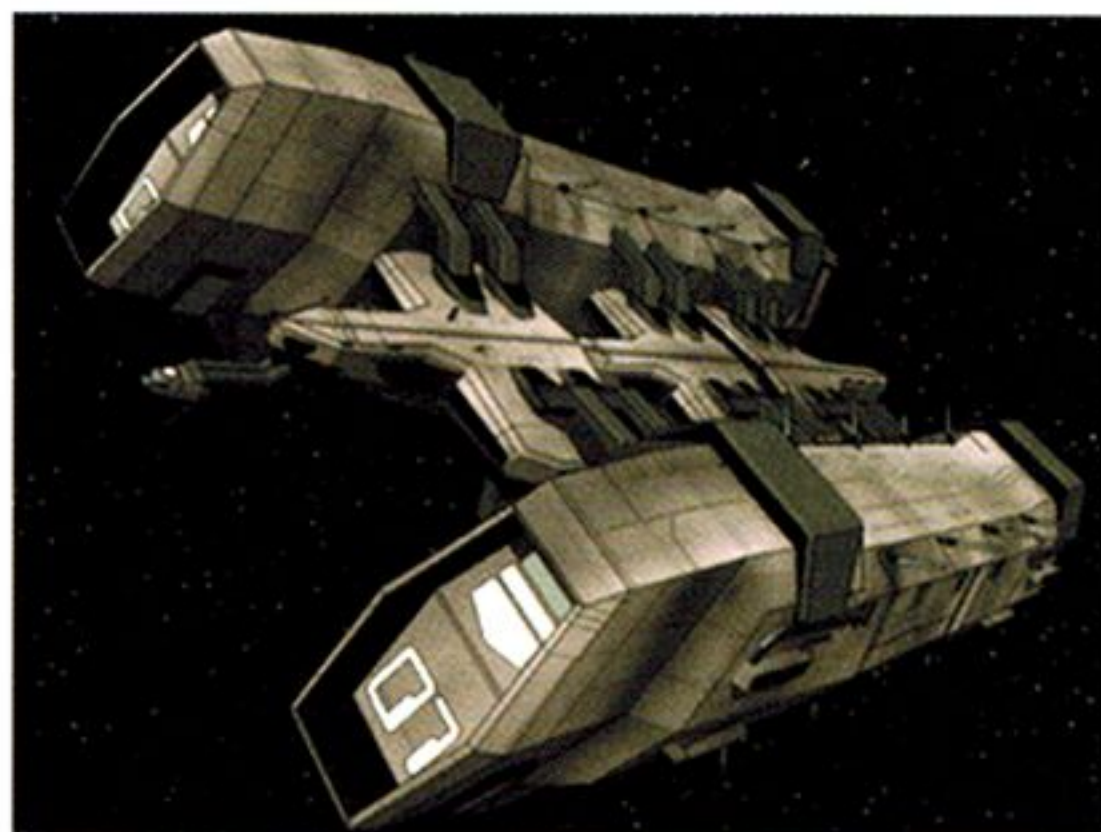
か：いえ。**こんなこともあろうかと，**数カット作っておきました

このカット（画像3）は，渡辺さんが，「やっぱ，オープニングって毎回流れるし，おいしいよね」

画像3　渡辺さんが「こんなこともあろうかと」作成しておいたオープニング用カット

画像4　渡辺さんオリジナルデザインのロストシップ「デスクラウド」

画像5　ボツになったデスクラウド初号機。監督はこれでもいいとはいってくれたが

画像6　デスクラウドの内部解剖図。もちろんデタラメ。渡辺さんの愛を感じる

画像7　なぜかデスクラウドたちが大量に出現するオープニング用カット

画像8　移民船メイフラワー。もともと輸送船としてデザインしたので無理がある

とかいって，密かに制作したものです。ソードブレイカーを制作したあと，次のモデリングに入りたいところですが，資料が届かない。その暇つぶしというわけです。

なお，DoGA内部でオープニングのフルCGバージョンを作るとか，アイキャッチを作ろうという企画もありましたが，さすがに実現しませんでした。

デスクラウド

「ロストユニバース」には，鈴木勤氏というプロのメカデザイナーがいます。しかし渡辺さんも，メカデザインは得意とするところです。自分がデザインしたメカがテレビアニメに登場するって，メカデザインをしている人には夢ですよね。そこで，せっかくだから記念に1体ぐらい渡辺さんにメカデザインをしてもらおうということになりました。

そこで選ばれたのが，敵のロストシップの中ではいちばん弱く，2話程度しか出てこない「デスクラウド」(画像4)です。デスクラウドは，地中に埋まっているところを盗賊に発見され，修理改造されて海賊船として使用されるという設定でした。しかし，デザインがストーリーなどに影響する点は特にないので，「おまかせです。ご自由にデザインしてください」とのこと。

しかし，このデスクラウドは一度ボツになりました。画像5がボツデザインです。宇宙船というより無人戦闘機のようで，大きさが感じられないというのがその理由です。とはいっても実は，監督や制作会社からはOKが出ていたのです。DoGAが内部的に「いや，渡辺さんはもっといいデザインができるはずだ」といってボツにしました。クライアントがOKといえば普通OKなのですが，この辺がこだわりというか，単に遊び半分でやっているのか。

ちなみに画像6は，渡辺さん直筆のデスクラウドの裏設定図です。足跡あたりに作者の愛情を感じるでしょう(なんやねん，宇宙船の足跡って)。

とはいえ，このボツデザインも結局テレビには登場することになりました。

監督：オープニング用に，敵がうじゃうじゃいる中をすり抜けながら，片っ端から撃墜するってカットをぜひ作ってください
かまた：でも，ソードブレイカー以外のメカ資料がまだひとつも届いていないですけど

ということで，オープニングにデスクラウド2種が大量に登場するカットが作られました(画像7)。ストーリー上，デスクラウドはこの世に1体しか存在しないはずですから，変なカットです。また，このころは色指定が決まっていなかったので，本編に出てくるデスクラウドとは色が違います。まぁ，そんな細かいところを気にする人はいませんか。

メカデザインの裏舞台

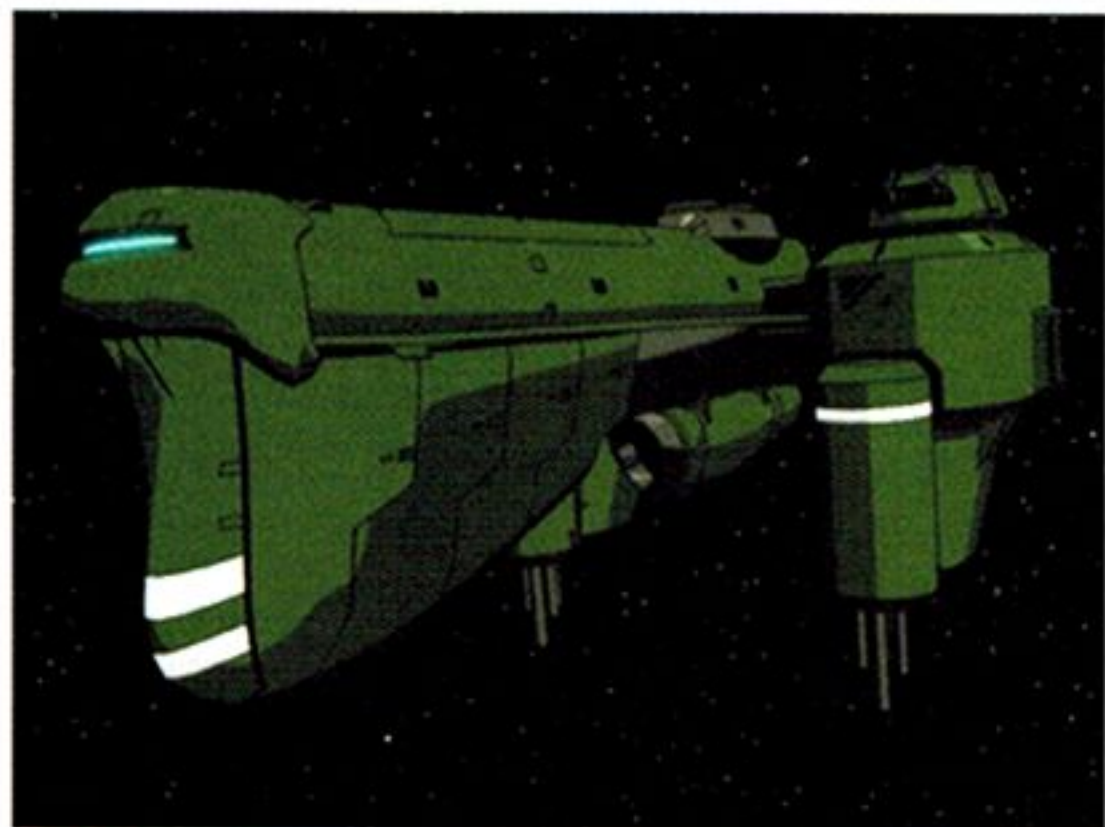
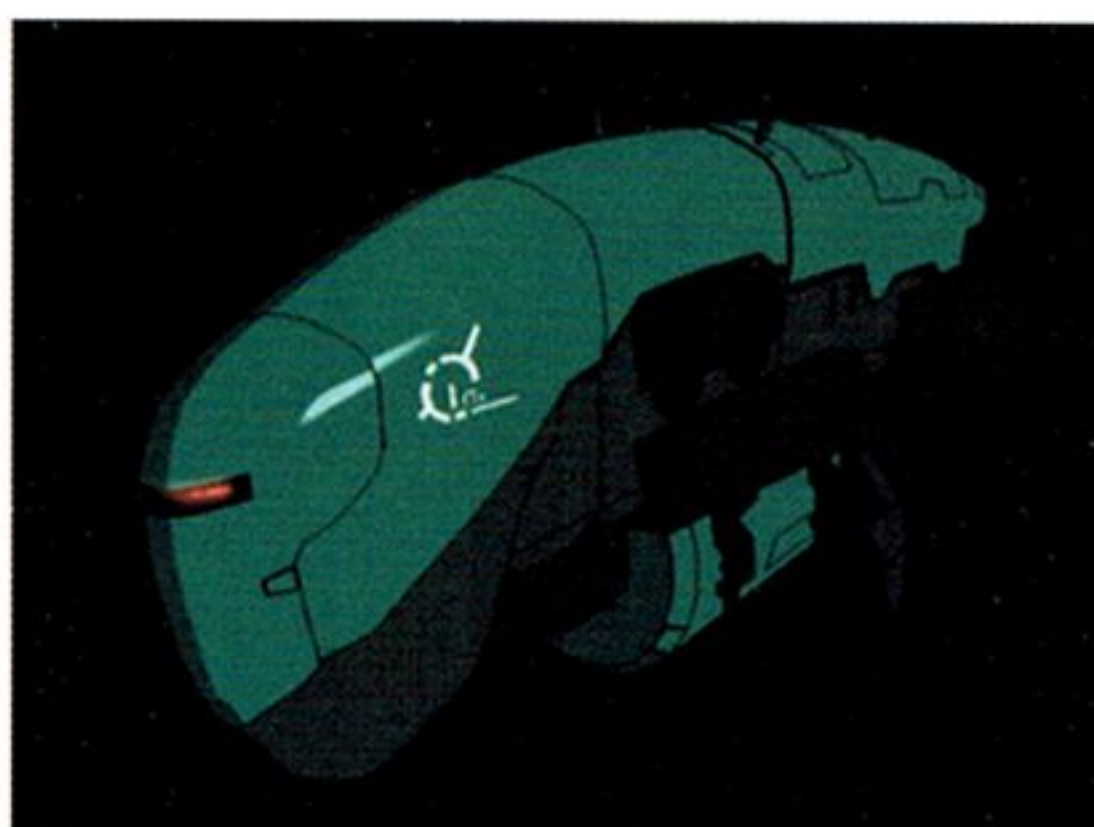
渡辺さんがメカデザインするのは1体だけの予定でしたが，再び依頼がきました。制作会社のほうでもデスクラウドが好評だったようです。というより，単にメカデザインが間にあわないということでしょう。

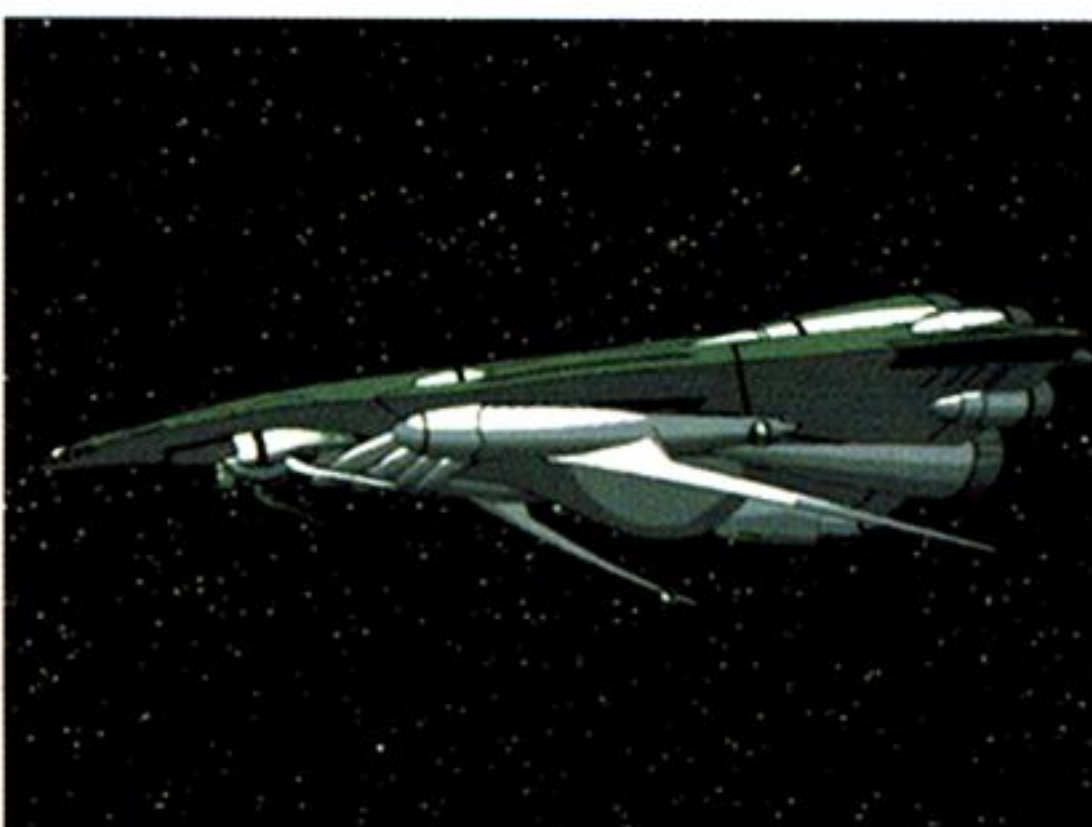
本職の鈴木氏のほうは，どうなっているのかと聞くと，「ほかのテレビアニメの仕事が忙しく，こっちの仕事はあまりやってもらえない」とのこと。は？　契約して仕事を引き受けた以上，プロとしてそれはあまりに無責任ではないか……と思いきや，別に契約なんかしていないそうです。単なる口約束で，お願いしただけ。条件なども決めていないとか。こんなのは業界ではごく一般的なんだそうです。……デタラメといっても，度がすぎる世界だと思いません？

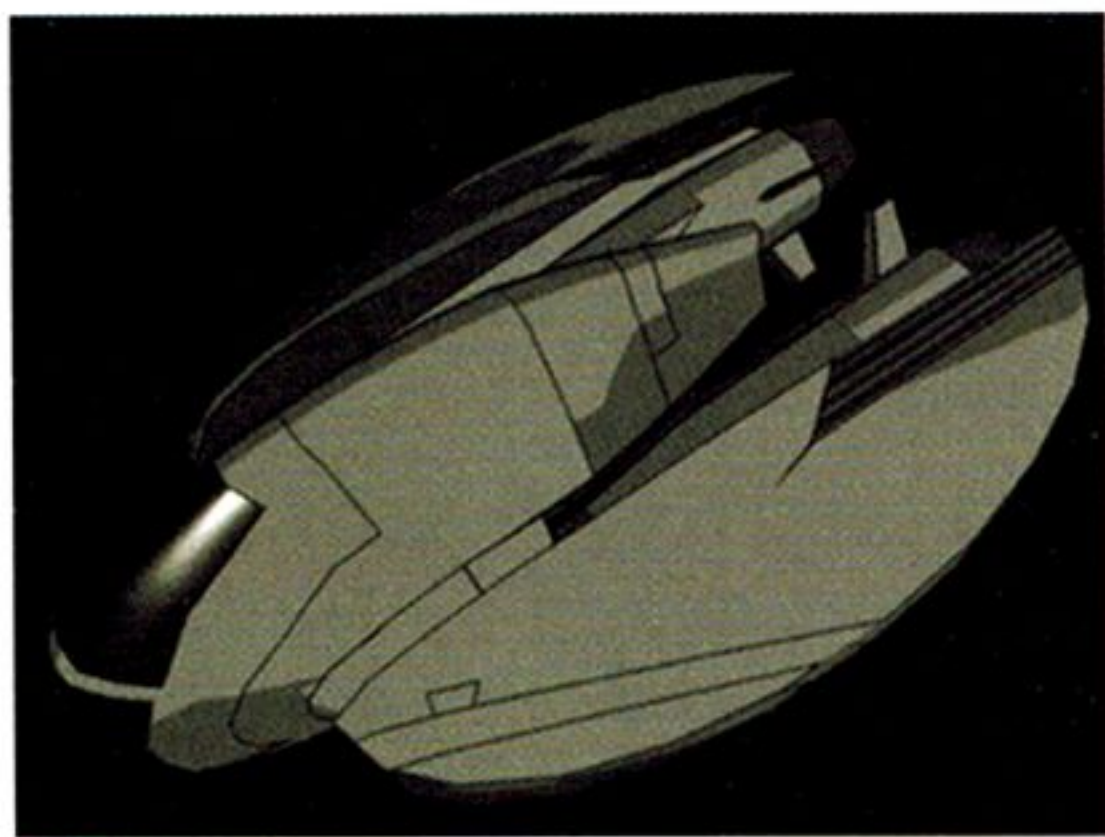
ということで，登場したのがメイフラワー号(画像8)ですが，これはちょっと失敗でした。このメイフラワー号が出てくる第10話は，もともとCGの出番がない予定でした。素直にお休みすればいいのですが，それではつまらない。移民船をモデリングして，CGパートを作ろうと提案したのです。もっとも，さすがにいまからモデリングしていると間にあわないので，渡辺さんが別の目的で作っていた輸送船をそのまま流用することにしました。

渡辺さんがデザインした宇宙船や戦闘機の数々

画像9　武器商人の船 エチゴーヤ

画像10　砲撃艦ウッタレヤー

画像11　輸送艦 ハコブンヤー

画像12　ジルの戦闘機

画像13　ロストシップ ラグドメゼキス

画像14　ロストシップ ネザード

画像15　DOGA-L2で作成された脱出ポッド。一瞬しか出ないので色もついてない

ところがよくストーリーを読むと，この移民船はスペースコロニーのように巨大で，中に住んでいる人たちが，艦橋の位置を忘れてしまい，制御もできないという設定でした。でも，デザインのほうはもともと輸送船なので，どう見ても普通の宇宙船より少し大きいといった程度にしか見えません。しかし，制作会社のほうもよっぽど切羽詰まっていたのでしょう。なにごともなくこのデザインでOKになりました。まぁ，なんともいい加減な話です。

以後，対外的には「メカデザイン 鈴木 勤」のままだったのですが，実際には渡辺さんのデザインが多くなりました。また，この頃からDoGAでネーミングするようにもなってきました(例外あり)。

- 武器商人の船　ワルソーネ級 エチゴーヤ：画像9
- 砲撃艦 ワルソーネ級 ウッタレヤー：画像10
- 輸送艦 ワルソーネ級 ハコブンヤー：画像11
- ジルの戦闘機：画像12
- ロストシップ ラグドメゼキス：画像13
- ロストシップ ネザード：画像14

次々にデザインが採用された渡辺さんは得意満面かといえばとんでもありません。「ボクなんかのデザインが使われて，本当にいいんやろか。あんなのが放映されて，鈴木さんは怒ってると思う」とビクビクしていました。

DOGA-L2の使用

放送開始直後から，これらのCGはDOGA-L2(当チームが開発した，初心者向けCG学習ソフト)で制作されているというウワサが流れていました。とんでもない間違いです。

想像するに……,
「CGは，DoGAが制作しているらしい」
「CGは，DoGAで制作しているそうだ」
「CGは，DOGA-L1，L2で制作している」
といった伝言ゲームが行われたのでしょう。しかし実はほんのちょっとだけですが，L2が使われているのです。

第15話で，悪王の船が爆発するとき，その悪王の爺さんが逃げ出す脱出ポットは，L2でデザイン&モデリングされました。なにしろほんの一瞬で，形状もよく見えないようなメカです。監督からの指示も，「どんなのでも結構です。丸にチョンでも結構です」ということでした。こんなメカのために鈴木氏や渡辺さんの手を煩わせることもありません。私がサクサクっと作りました(画像15)。

L2ユーザーなら，どのパーツをどこに使ったかすぐわかるでしょう。やっぱり，L2は作業が早い！ 構想からモデリング終了まで約15分です。名前も安直に「スッタ・コーラ」となりました。

その他のカットでも，宇宙に漂う残骸など，何度かL2を活用しています。これで，L2もテレビアニメ制作にも使われた本格的CGアニメ制作ソフト"という肩書きがつきます。う～ん，ちょっと無理があるか。

評価向上のための努力

今回のお仕事に対して当方ではひとつの目標を決めていました。それは「CGパートは遅らせない」ということです。アニメ業界では，どのパートもスケジュールより遅れることが半ば当然のように行われています。それがほかのパートに悪影響を及ぼし，さらに遅れる要因となります。そんななかで，CGパートだけは，全26話，納品を1日たりとも遅れないようにがんばり，CG業界に対する信頼を勝ち取ろうと考えたわけです。

その結果，締め切りはなんとか死守できたと思います。たとえば，絵コンテが大幅に遅れて納品日が変更されたとか，厳密には遅れなかったと断言できないケースもあります。でも，ほかのパートと比べればきわめて優秀だったことは間違いありません。

とはいえ，むしろほかのパートより先行しすぎて，足並を乱していた感があります。たとえば，次週の予告編は全部CGというケースが何度もありました。CGは仕上がっているのに，セルがまったく追いついていないのでしょう。

それからもうひとつの努力としては，多くのカットで，複数パターン納入しました。たとえば特殊効果の使い方にバリエーションを設けたり，絵コンテどおりに作ったカットと，我々ならこうすると思いどおりに制作したカットを両方制作し，好きなほうを使ってもらうというわけです。もちろん，こんなことをするとロスが多いのですが，なにしろセルパートのほうが遙かに忙しいので，細かい問い合わせをして，手を煩わせることはできませんでした。それに，返答を待っていると，その時間のほうがロスになります。

おかげさまで，納品したカットに対して，NGはひとつも出ませんでした(下記のトラブルを除く)。監督が遠慮している面もあるのか，いつ問い合わせてもゼンゼン問題ない"といわれました。

かまた：もう少し，どこが悪いとか，こうしてほしいといった要望を返してもらわないと，こちらもスキルアップのしようがないんですが
監督：いや，ごもっとも
かまた：どこか問題はないでしょうか
監督：そうですねぇ。う～ん。う～む。いや，ゼンゼン問題ないです。OKです

ポケモンNG

第11話で，最初で最後のNGが出ました。"ポケモン規制"です。

一昨年，テレビアニメ「ポケットモンスター」の画面を見ていた子供たちがてんかんの発作を引き起こす事件がありました。これ以後，再びこのような事件が起こらないようにとテレビアニメの表現方法に厳しい規制ができてしまいました。これがポケモン規制です。たとえばテレビ東京の場合，画面の10％以上の面積が，1秒間に3回以上点滅してはいけないなど，こと細かに記した「アニメ制作ガイドライン」全9ページが作成されています。また，テレビアニメの冒頭に「アニメを見るときは部屋を明るくし……」といった注意が表示されるようになりました。

今回の場合，ソードブレイカーが集中砲火を浴びるカットで，砲弾の炸裂が激しすぎて，結果的に画面が点滅しているというのです。個人的には，あれでてんかんを引き起こすとは到底思えません。とはいえ，局の意向に逆らえるわけもなく，結局，砲撃の色を変えたり，画面全体の明るさを落とすといった方法で対応し，納品し直しました(画像16)。

しかしながら，このNGについては，当方の責任ではないと考えています。当時アニメ業界がポケモン問題に敏感になっているのは十分承知していました。ですからこのカットでも，本格的に制作に入る前にサンプル映像を作り，このような表現を行うつもりだが問題はないかと，わざわざ確認を取っていたのです。そして，テレビ局側から問題なしという回答を得たうえで制作しました。

それなのに，なぜNGになったのか？　それはテレビ局側のチェックの担当者が変わったからだそうです。つまり，ガイドラインが作成されているにも関わらず，最終的にOKかNGかは，担当者の主観的判断，気分次第というのです。これは困ります。

また，テレビアニメだけ目の敵にするのも変な話です。てんかんは，特定の色の点滅に対して起こるもので，別にテレビアニメに限ったことではありません。実写のテレビドラマでも起こるものですし，その辺の警報ランプの点滅でも起こりえます(映画「アンドロメダ病原体」のなかでもその危険性が描かれている)。特に，CMなんかでは，点滅など多用しているのが現状です。

あるテレビアニメでは，吹雪のシーンで，画面上を横切る雪が白の点滅になっているといってNGになったとか。では，どうやって吹雪を表現しろというのでしょうねぇ。

CGの効用

最後にテレビアニメにCGを用いることの効用を考えてみましょう。まずコストですが，これは結構よかったはずです。もとから1話につき30秒分でペイする制作費しかもらっていません。それが後半になると，毎話1分以上制作していたのですから，単純に計算すれば，コストはセルの半分

画像16
ポケモン規制によって変更された画像。上がオリジナル版。下が修正版

以下ということになります。

特に戦闘シーンでは，ビームや爆発などの透過光処理が必要になってきます。このようなカットをセルで制作すると，通常よりセルの枚数も多くなる部分ですし，特殊な撮影をするため手間もかかります。この辺をCG化したおかげで，ロストユニバースのセルの総枚数は通常より3割がた少なくなったそうです。ということで，CGは決して高価ではなく，むしろセルより安くできるといえるでしょう。

もっとも，逆にDoGA側の採算はかなり苦しいといわざるをえませんでした。採算の計算なんて単純にできるものではありませんが，大ざっぱにいえばあと3割ぐらい増やしてほしかったと思います。制作会社の社長さんは，「テレビアニメの仕事をすれば，知名度がガガっと上がります。今回は赤字でも，次からよい仕事がバンバンきますよ」とおっしゃっていました。確かに知名度は結構上がったと感じるのですが，特に仕事が増えたとは感じられません。あれは社交辞令だったのでしょうか。

またクオリティの点でも，CGパートは概ね好評を得ています。特に監督には喜んでもらえました。やっぱり宇宙船などのメカに限っていえば，セルよりCGのほうが優れているといえるでしょう。セルのアニメの途中にCGを混ぜると，まだまだ違和感があるとはいえ，セルシェーダーや独自技術によって，通常のCGよりは遙かに違和感を少なくすることができました。これもひとつの成果だと思います。

このようにテレビアニメにおいてCGの使用は非常に有効といえるでしょう。だからといって「全部CGにしてしまえ」とは少しも思いません。CGが優れているのはメカのアクションぐらいで，人間の描写などは断然セルのほうが上です。だから，CGが得意なところだけCGを使うというやり方が当面主流になってくるでしょう。

今回の場合，予算の問題などがあって，宇宙船が出てくるところをすべてCGにすることができませんでした。そのため，CGで描かれた宇宙船と，セルの宇宙船が混在し，絵柄にばらつきが出て，見苦しかったように思います。宇宙船は一貫してCGで描くべきでした。しかし，たとえば地上に宇宙船が着陸していて，その宇宙船の前を主人公が横切るといったカットでは，セルとCGの合成が必要になってきます。これは単に合成の手間が増えるというだけでなく，制作の足並みを揃えることが重要です。地上の背景画，CGの宇宙船，主人公のセルという順番で仕上がらないと作業が進まなくなり，タイムロスが大きな問題になるでしょう。今後の課題といえるでしょう。

おわりに

本文を読み返すと，DoGAは遊んでばかりいたように思えますが，そんなことはありません。特に後半は，毎週1分以上，20カット近くをひとりで制作していたのですから，すごい作業量です。とはいえ，面白かったのも事実です。毎週，自分たちが作った作品がオンエアされるのは，なかなか楽しいものです。

その半面，テレビアニメに関しては素人同然の私たちが，長年下積みをしてきた方々をさしおいて，オープニングにデカデカと名前を載せ，クライマックスのおいしいところを制作するのは，心苦しいものを感じました。制作会社のスタッフのごく一部には，CG使用反対論があったそうですが，ごもっともなことだと思います。にも関らず，温かく迎えてくれた監督以下制作スタッフの方々には大変感謝しています。

それから心残りなのは，制作時間や予算が足りなかったことです。十分な人数のスタッフも割けず，不満の残るカットも作り直している余裕がありませんでした。やはり，宇宙船が出てくるすべてのカットをCG化できなかったのも残念です。

次は，劇場アニメもやってみたいし，ロボットアニメにも挑戦したいと思います。でも，ロボットアニメになるとコストがあうか疑問です。宇宙空間をバックに宇宙船が飛ぶのと，地上をロボットが走るのとでは，CG制作の手間がまったく違います。アニメ制作会社はそれを理解できないでしょう。少なくとも今回のような予算ではロボットはできません。

そう考えると，今回のスキルを生かしながら，宇宙船を100%CG化し，セルとの合成も多用するぐらいのところからチャレンジするのが妥当かもしれません。とはいっても，こちらがいくら希望しても，依頼がこなければラチがあきません。ということで，ただいまテレビアニメ制作のお仕事募集中です！

テレビアニメ「ロストユニバース」

監督：渡部 高志
原作：神坂 一
制作：EG FILM
テレビ東京系列 1998年4月～9月 全26話放送
ストーリー：主人公のケインが宇宙船ソードブレイカーに乗って，射撃の名手ミリィとコンピュータのキャナルとともに大活躍。ソードブレイカーは，前文明の遺産である「ロストシップ」で，圧倒的な力を持つ。ほかにもロストシップはいくつか発見されており，それらと宿命の戦いが繰り広げられる。

荻窪圭
Kei Ogikubo

QuickTimeVRの理論と実践

図1-01　被写体をレンズの右端に置いて撮影した写真の被写体部分をトリミングしたもの

図1-02　被写体をレンズの中央に置いて撮影した写真の被写体部分をトリミングしたもの

図1-03　被写体をレンズの右端に置いて撮影した写真の被写体部分をトリミングしたもの

360度のパノラマムービーなんてのは好事家の道楽かマルチメディアコンテンツの彩りかと思っていたら，どうも世間ではそうでもないらしい。カシオのQVシリーズがパノラマ撮影支援機能を持ったと思ったら，キヤノンのPowerShotも同様の撮影機能を持ち，それで終わりかと思ったら，VAIO君ちのC1までがパノラマ撮影ができるなんてのを宣伝文句のひとつにしてる。いやはや，妙な世の中である。

パノラマ写真ってのは銀塩の昔から結構ポピュラーな存在で，確かに画角が限られているカメラを手にしたら，広い範囲をひとつに写しちゃいたいと思うのも人情ってものだ。でもね，そんな甘いわけぢゃないのよ。

てなわけで，QuickTimeVR黎明期からあれやこれやと遊んでいる荻窪圭は，これをチャンスと見て，VRな話をすることにした。だって，こういう話を書かせてくれる雑誌ってほかにないんだもん。わははは。ではよろしく。

理想的な360度パノラマ撮影とはなにか

もしあなたがVAIO C1を持ってぐるぐるとその場で回ったり，カシオのQV-7000SXとか5500SXとか，さらにはキヤノンのPowerShot A5系のデジカメのパノラマ撮影モードを使ってパノラマ撮影をしようと考えているなら，それに見合った成果を得られるかどうか保証の限りではない。

3，4枚撮影して，画角90度分程度のパノラマ写真を作ってOK，というのであれば，さらにクオリティはどーでもいいというのであれば，なんとかなる。でも360度分作って遊ぼうと思うのなら，それは強靱なバランス感覚と身体能力を必要とするだろう。

たとえば，写真の端っこでぴったりと重なるように回転しながら2枚の写真を撮ってみて，それをひとつにつないでみるとよい。100%ぴったりにはつながらないのである。なぜか。目の前に直方体があったとする。それをレンズの中心にとらえて撮影したときと，レンズの左か右の端にとらえて撮影したときで，まったく同じように写るだろうか。

ちょっと汚くて申しわけないが，四角い物体がわかりやすいので，キヤノンのPowerShot A5をいままさにこの原稿を書いている机の上に置いてそのように撮影し，該当部分だけを切り出してみた。35mm相当のレンズで撮影したので極端には出てないが，レンズの端に置いた場合はそれに応じて歪んで写っているのがわかる。それぞれ，レンズの右端に被写体をとらえたとき，中央にとらえたとき，左端にとらえたときの写真だ（図1-01～03）。

レンズが広角であればあるほどこの傾向は顕著になる。だから，友達同士で横に並んで写真を撮ってもらうとき，両端には立たないほうがよい。このようにちょっと歪んで写るからだ。これを読んだ女の子はちゃんと注意しておくように。

まあ，レンズが広角でなければこういうのは目立たないからいいんだけどね。なぜ広角だと目立つかというと，画角が広いからである。まあこれだけでも端と端を綺麗につなぐのは困難だってことがわかるだろう。

実際にはこういう歪みは「パノラマ作成ソフト」の「ステッチ機能」が補正してくれる。補正してくれないソフトもあるが，そういうのは使わないほうがよい。実例はあとで挙げよう。

さらに，手で持ったカメラで「常に水平を保って撮影する」ことは，よほど強靱なバランス感覚と身体能力を持っていても困難だ。画像が左右に傾いているときは，実をいうと，なんとでもなる。適当なフォトレタッチソフトで傾きを直してやればいいからだ。

問題は，前後に傾いてしまうことだ。こればっかはいくらファインダを覗いてもわからない。前後の傾きってのはどうしても生じるからだ。人間ってのは「構図の中にある目立つものを中心に入れたがる」という性質があり，カメラを持つ高さや左右の水平をどれだけ慎重にあわせても，気がつくとカメラが前後に傾いてしまうのだ。もし地平線や水平線がくっきり見えている場所で撮るならそれを目安にできるので問題ないが，そんな場所は日本には（たぶん）ない。

そうした前後の傾きを補正するのはすごく困難だ。微妙な遠近となって表れるからである。

一般にパノラマを撮影するときは「カメラを水平に保ち」「等間隔で撮影すること」が大事といわれるが，実をいうと，それはかなり優先順位が低いことなのである。左右の傾きはフォトレタッチソフトでなんとかなるし，等間隔でなくても優れた「パノラマ作成ソフト」ならなんとかしてくれる。でも前後の傾きはいかんともしがたいのだ。

これらをクリアしても，まだ難関が残っている。回転の中心はどこ？　という問題だ。手で持って

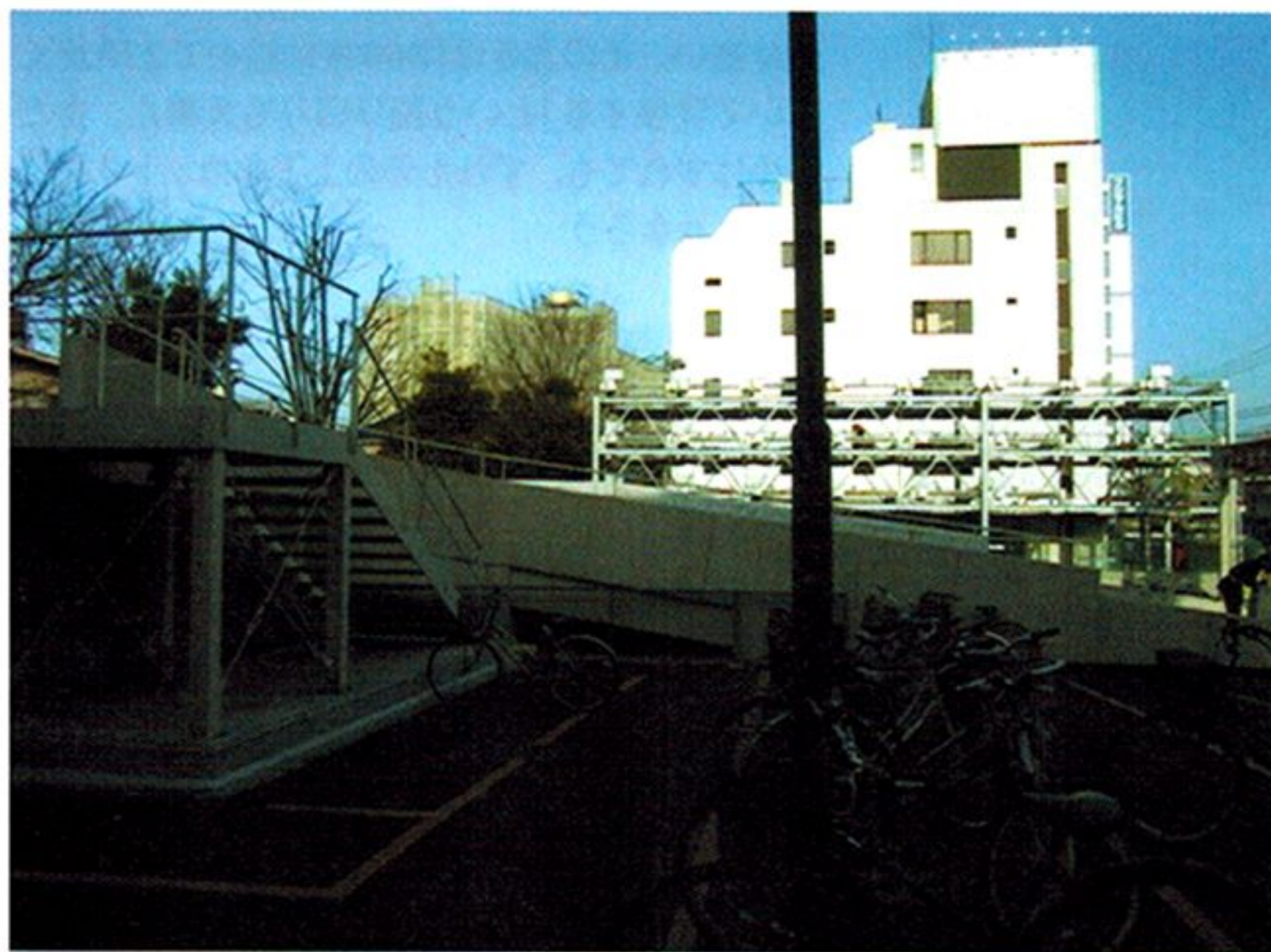



図2-01

図2-02

図2-01，02　カメラを手に持ち，パノラマ撮影モードにして適当に撮ったうちの2枚。街灯とその向こうにあるビルの位置関係に注目。窓の位置を見ると距離の離れた被写体感のズレがよくわかる

撮影するとき，回転の中心は自分の身体となる。当たり前だけど，そうなのだ。でも，論理的には，回転の中心はカメラでなければならないのである。もっと厳密にいえば，回転の中心は「焦点」でなければならないのだ。

これは一般においてかなり軽視されている問題だ。確かに軽視しても構わないことはある。回転の中心がずれていて困るのは，構図の中に「遠いもの」と「近いもの」が混在しているときだけだからだ。回転の中心がずれていると，遠いものと近いものの位置関係が大きく狂ってしまうのである(図2-01，02)。

これはPowerShot A5 zoomのパノラマ撮影支援機能を使い，カメラを手で持って撮ったものだ。遠くのビルと街灯の位置関係を見てもらいたい。これだけずれちゃうと補正は難しい。でも室内や町中で撮ろうと思ったら，このくらいはざら。

ではどうするか。カメラってのは原理として，レンズから入ってきた光が交差し，反転して受光面(デジカメならCCD)にたどり着く。その交差する点を焦点という。この焦点を中心にして回転させねばならないのである。でも，カメラのレンズは何枚ものレンズで構成されているので，外から見てどこが焦点なのか，なんてまったくわからない。特にデジカメだとなおさら。

だから，簡易的にはいちばん外側にあるレンズの中心と回転軸をあわせておくことが推奨されている。厳密にやろうと思ったら，たとえば，図2のような状況の場合，街灯がフレームの左端にあるときでも右端にあるときでも後ろのビルとの関係がずれないような位置を見つければいい(らしい)。って，それはあまりにめんどくさいので(そもそも，手持ちのカメラでは不可能だし)，やったことはないけど。

で，簡易的には図3のような感じである。これはCOOLPIX910+ワイコンを，マンフロット社製のパノラマ撮影専用雲台に取り付けた場合の写真だ(図3)。

以上のような点に気をつけ，できれば常に半分くらい重なるよう360度分撮影するのがベターだ。24mmのレンズで縦位置で撮影する場合，18枚くらい撮ればよい。そのくらい撮っておけば，途中でどうしても使えない写真が交じっても強引につなげられるからだ。めんどーなときは12枚くらい撮っておけばなんとかなるけど，あまりすすめない。

ふー，疲れた。

でもこれで完璧か，というとそうではない。フォーカスと露出の問題が残っている。屋外であれば，太陽の位置がある。当然，日陰が多ければ露出は高くなるし，日向なら抑えられる。でも，撮影する方向によって明るさが違うと，つないだとき不自然になる。フォーカスについてはわかりやすいだろう。一般に広角レンズのほうが被写界深度は深くなるが，それでも，あるカットでは背景の風景にピントがあっており，あるカットでは手前の電柱にピントがあっていた，では困る。

さらにデジカメだと「ホワイトバランス」の問題がある。カットによって色あいが違っては困るからだ。多少の露出のずれはフォトレタッチソフトでごまかせても，色のずれをごまかすのは難しい。必ずホワイトバランスは太陽光かタングステン灯か蛍光灯に固定してから撮影するべし。

ここまでクリアすればほぼ完璧だ。

だがしかし，最後に，どれだけ気をつけて撮影してもどうしようもない問題がある。無人の観光地や無人の建物の中でもない限り，絶対「動いている人や犬や猫やクルマや自転車」があるのだ。当然，あるカットには写っていて別のカットには写ってないってことは日常茶飯事である。それがつなぎ目で起きたら……。それはどうしようもないし(ただ単に幽霊ができるだけだし)，そんなことに気を取られていたら撮影可能な場所なんて限られてしまうので，もし気になるなら「ステッチが終わったらフォトレタッチソフトでごまかす」のがおすすめ。

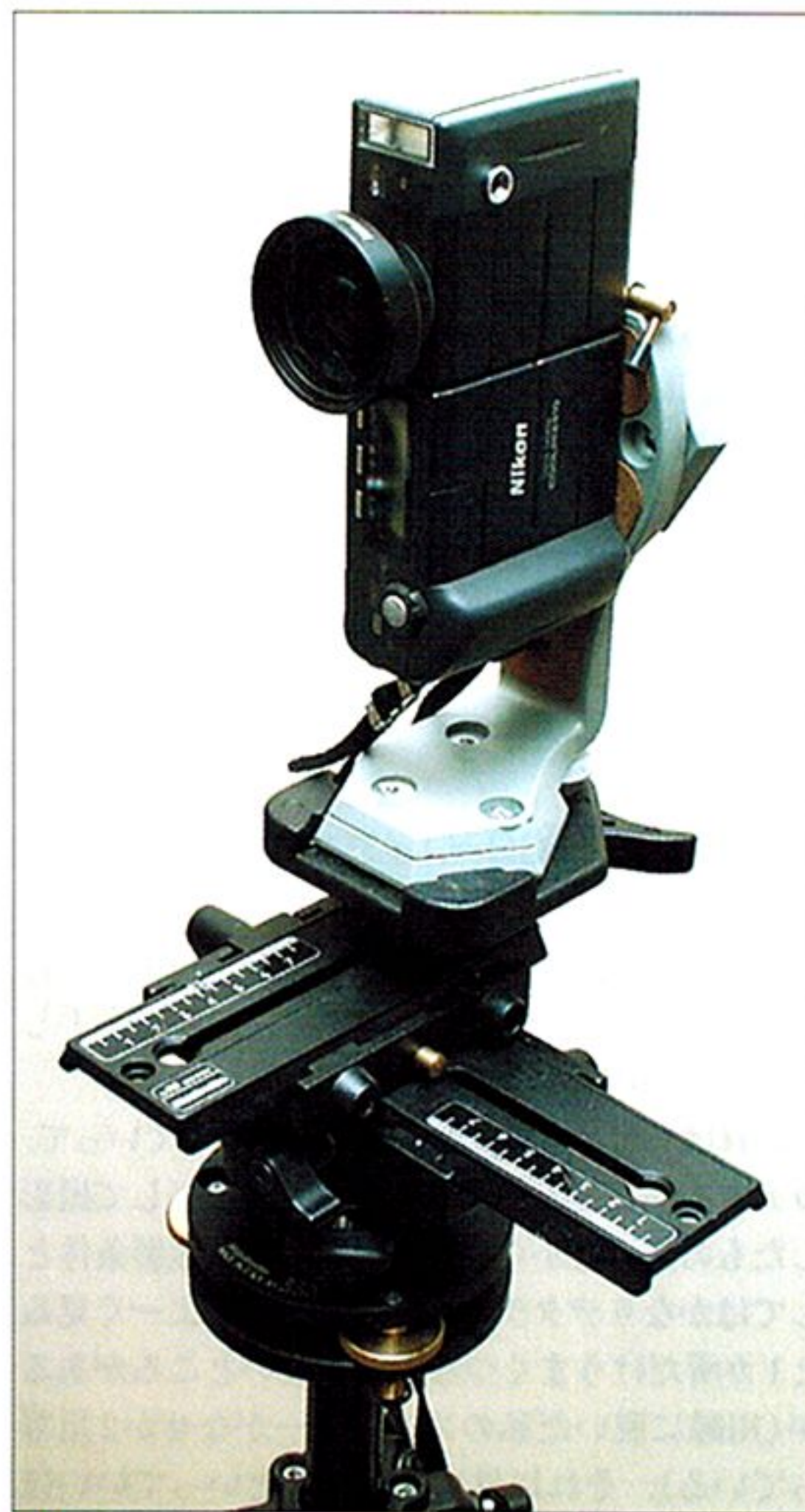



図3　これくらいの装備があればパノラマ写真もそれなりに撮れる

ステッチというのは，複数の連続した画像をつなぎあわせる作業のことで，パノラマ作成ソフトの根幹をなす大事な機能である。

撮影の実践

なーんて書いたけど，実際にはそこまで気をつけて完璧に撮影する必要なんてないよーん。わははは。すまぬ。まあ，理論と実践を一致させるのは困難だってことだ。

たとえば，さほど広くないの部屋の中心にカメラを置いて，超広角な(15mmとか18mmとか)レンズでつなぎ目が完璧な360度パノラマを作ろう

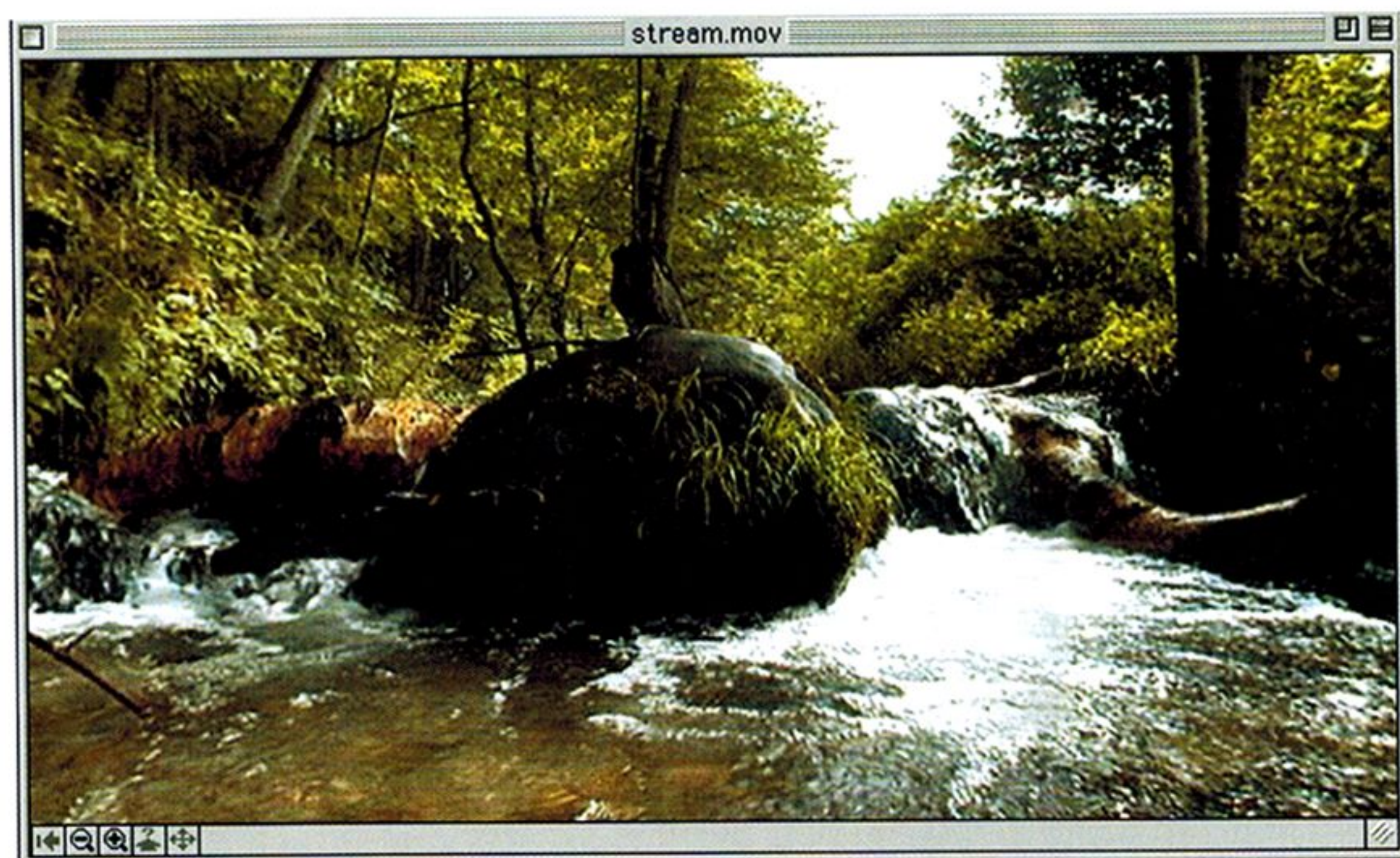



図4　山の中の小川に入っていって撮影した無謀なVR。寒かった。9月に長野の山中で撮影

と思ったら，かなり完璧に撮影しなければならない。それがいちばん過酷な条件だ。ある程度は「優れたステッチソフト」や「フォトレタッチ」で補正できるが，限界はある。

逆に，ほとんどの被写体が無限遠にある「見晴らし台の上」とか「山の上」とかで標準レンズ（35mm～50m程度）で撮影するなら，余計なことはほとんど気遣わなくてよい。これはもっとも歪みが出ないケースだからだ。こういう場合は手持ちでも十分である。

では作例をひとつ見せよう。

1.COOLPIX900+ワイコン+水準器で手持ち撮影する場合

CD-ROMに入っているstream.movである。要QuickTime 3.0なので持ってない人は入手してください（図4）。

これは，川の中にばしゃばしゃと入っていって，カメラを手で持ってぐるっと360度回転して撮影したものだ。だから360度パノラマの撮影条件としてはかなりデタラメである。でも，よーく見ると1カ所だけうまくつながってないところがあるが（川縁に脱いだ私のスニーカーがなぜか2足写っている），それ以外はまったくといっていいほどわからないはずだ。

このときは次のような手順で撮影した。

①川の中で中腰になり，腰のあたりにカメラを構え，COOLPIX900のレンズを回転させて，液晶モニタを正面から見られるようにする。
②レンズ部分はほぼフラットなので，そこに水準器を置く
③水準器を見て水平を確保しながら，液晶モニタでモニタリングしつつ，半分くらい重なるように撮影していく
④足から冷えてくるのに耐え，撮り終えるまでおしっこを我慢する

である。

ちなみに，ホワイトバランスは太陽光に，フォーカスは遠景に固定したが，露出はオートのままだ。COOLPIX900は露出を固定することもできるが，山の中の小川なんて撮影する方向によって輝度差が激しいので，固定しちゃうとある方向を見たときは暗くて，別の方向を見たときは真っ白，って事態になりかねないのだ。だから，オートのままで撮影して，あとはフォトレタッチソフトで補正しながら作っていくほうがよいと判断した。

もちろん，回転の中心がカメラではなく自分の身体になっているので，撮影した写真をひとつひとつ見ると，大きなズレが生じている。だが，それはステッチソフト（アップル社のQuickTime VR Authoring Studio。略称：QTVRASを使った）が判断してズレをごまかしてつないでくれたのだ。優秀なソフトウェアさま，ありがとう。

2.普通のデジカメで普通に撮影する場合

COOLPX900と910は上記のようにレンズ部分が回転するうえに，ほぼレンズ部がほぼフラットなので水準器も載せられ，純正のワイドコンバータがあるため，パノラマ撮影には非常に向いている。

でも，そうじゃないカメラの人はどうするか。コツがある。デジカメの場合（最近，デジカメしか使ってないので，私が「カメラ」といえば「デジカメ」を指すと思ってください），

①液晶モニタをonにする
②カメラを目の高さに構え，液晶モニタに写っているもののなかで「目の高さにあると思われるものが常に中心にくるようにする」。これが大事だ。たとえば，遠くに人が写っていれば，その人の頭の位置が上下の中心にくるようにしよう。ある程度のずれはしゃあないが，これに気をつけていれば大きくハズれることはないはず。そうすることでカメラが常に正面を向いてくれるように保つわけだ
③前に撮ったカットの構図を忘れないようにし，常に50％以上は重ねて撮るつもりでたくさん撮影する。60％以上重ねるつもりでもいい
④それでも，最初と最後で大きくずれている可能性は高い。そのときは「360度は諦めて270度くらいで我慢する」という割り切りも大事だ。ただ多少はずれても，ずれた分の上下をカットすればなんとかなろう

3.三脚を用意できる場合

三脚があれば簡単である，ように見える。確かに撮影は簡単だが，思ったほど綺麗には撮れないかもしれない。ほとんどのデジカメは三脚穴がレンズの中央にあってないからだ。ましてや，回転軸と焦点をあわせるなんて至難の業。手持ちよりはずっとましだが，撮影する場所によっては思ったような出来にならないことがある。多少の覚悟は必要だ。

4.予算と体力がある場合～その1～

もっとクオリティの高いパノラマ撮影をしたいときは，自分でQuickTimeVR撮影用の雲台を作ってしまうのもいい。

まず，三脚の雲台を外す。そして，三脚の水平をあわせるレベルアジャスタ，雲台を回転させるパノラマアジャスタ，カメラの位置をずらしてレンズの中心と回転軸をあわせるための「スライダー」or「シフター」（マクロ撮影用の機材として売っているものが便利），そして最後はカメラを縦位置にする「アングルチェンジャ」である。

全部揃えると非常にややこしくて金もかかるので，おすすめはしないが，1つひとつの役割がしっかりしてるので，お勉強にはなる。

レベルアジャスタで水平をあわせれば，三脚自体が多少斜めになっていても，レベルアジャスタより上にある部分の水平は保たれるという意味で使う。

パノラマアジャスタは，角度を測りながら雲台を回転させるための道具であって，ちゃんと回転させられればなんでもよい。どのくらい回転させるかは，計算して割り出しておくと撮影後の処理が楽になるし，液晶モニタを見ながらアバウトに回転させると，撮影後の処理の際面倒なことになる可能性があるってだけだ。

スライダーやシフターと呼ばれるパーツの重要度はかなり高い。カメラにもよるが，レンズの中心と三脚穴が大きくずれていると，当然三脚の回転軸とレンズの中心がずれてしまってよくないからだ。マクロ撮影用のスライダーはそういうズレを解消するのに向いている。X-Y軸の両方向に動かせるものが便利だが，かさばるのが難点。

最後のアングルチェンジャはちょっと難関。よいものがなかなか見つからないからだ。これがあればカメラを簡単に縦位置にセットできる（図5）。

なぜ縦位置にするかというと，そのほうが上下の画角が稼げるから。左右の画角は撮影枚数を増やせばなんとでもなるが，上下はそうはいかない。そして上下の画角があればあるほど，360度の風景を回転させたときの臨場感は大きくなるし，写し込まれる範囲が広くなるのでより広くを見渡せるわけだ。

なお，上に挙げたものを全部揃えなくても，必

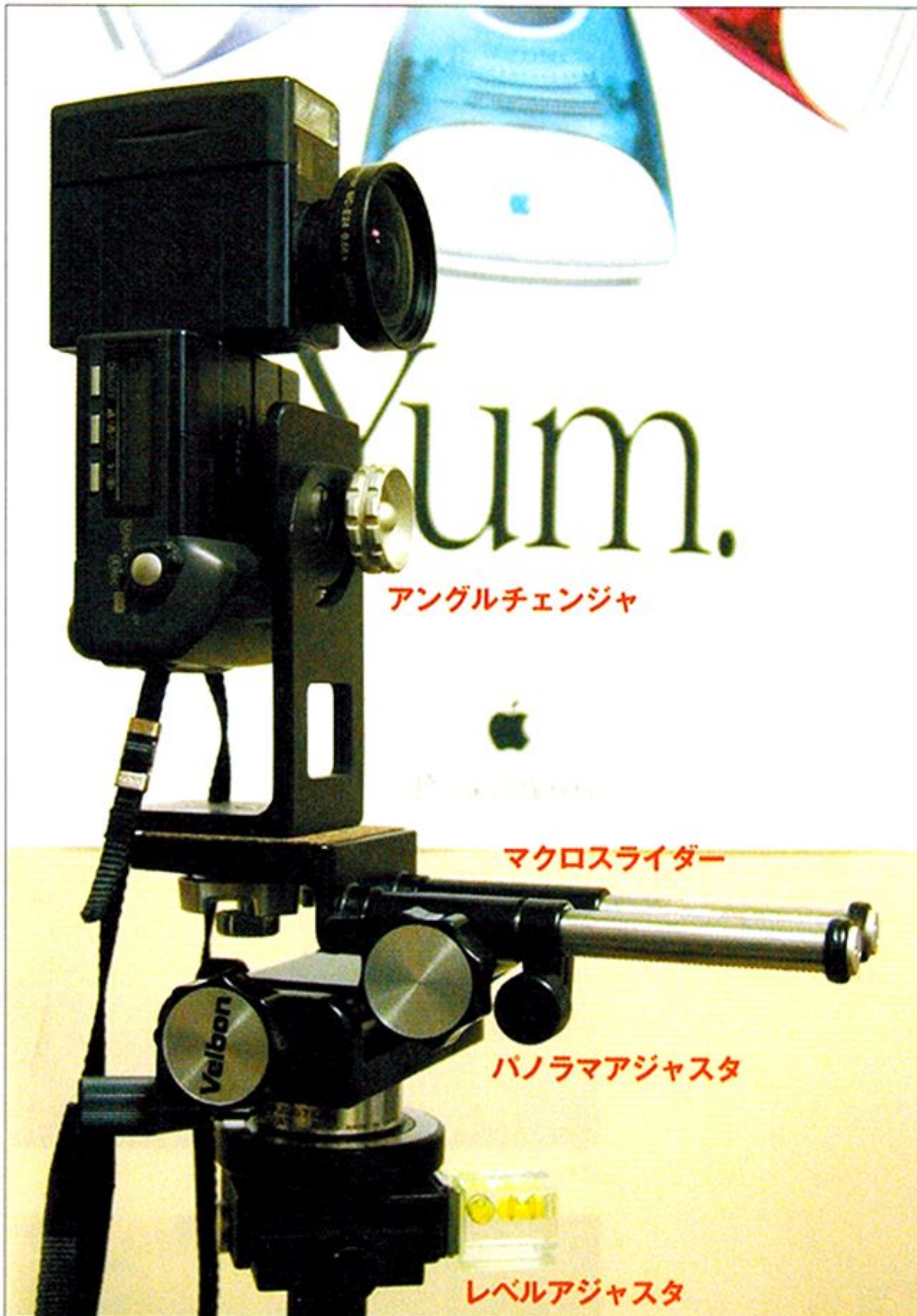



図5　荻窪圭特製VR撮影三脚セット。上から，アングルチェンジャ，マクロスライダー，パノラマアジャスタ，レベルアジャスタと積み重なっているが，ここまでやる必要はない。全部足すとけっこうな金額である（特にアングルチェンジャが高い）

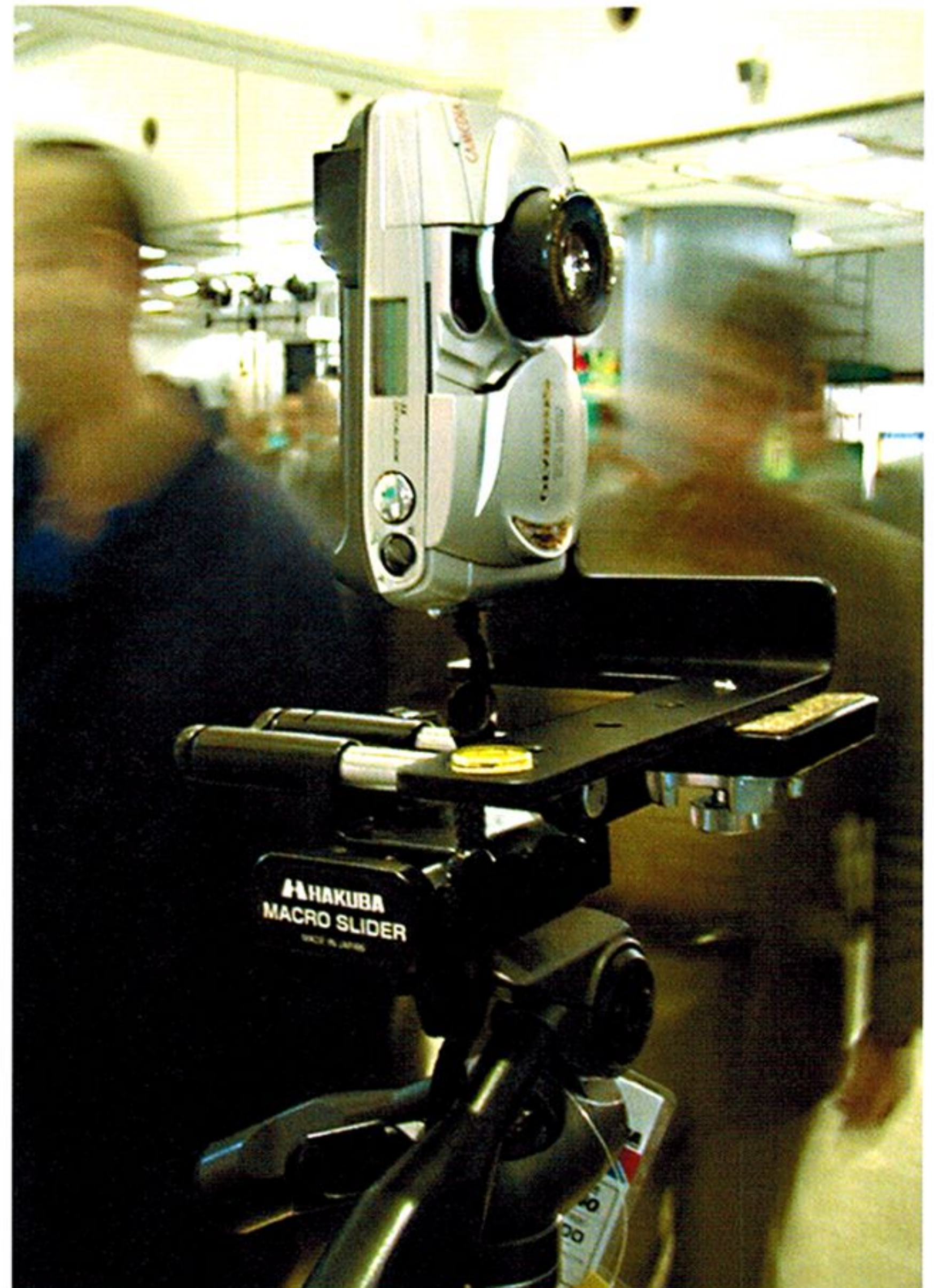



図6　カメラショー'99で見つけたパノラマVR撮影用の雲台。カメラを最適の位置で縦に固定するアダプタがなかなかないので，早く発売してほしい

要に応じてひとつか2つ用意しておくだけでも撮影は随分容易になるので試してみる価値はあるだろう。

5.もっと予算と体力がある場合

アメリカではKaidan社がKiWiという360度パノラマ撮影用の雲台を出しているが，日本で入手するにはオンライン販売を使って取り寄せるしかない。

日本で入手できるのは，イタリアの有名な三脚ブランドである「マンフロット社」が出しているパノラマ撮影用の雲台であろう。「本庄」というマンフロット社の代理店が扱っているはずだ。ただし，8万円くらいする。図3の写真がそれだ。

5月くらいまで待てるなら，日本の写真用品メーカーであるハクバがパノラマ撮影用の雲台を発売する予定ということだ。カメラショー'99で見つけた。発売日・価格ともに未定だが，Windows用のステッチソフトとセットで販売されるらしい。これなら2～3万円で買えるのではないかと思う（図6）。

6.もっとも楽な撮影方法は

と，スチルカメラを使う方法を述べてきたが，もっと楽をしたい場合は，ビデオカメラって手もある。ワイド側にしたビデオカメラを持ってぐるぐる回りながら撮影し，それをパソコンでキャプチャして，使えるところだけ引っ張ってくればいいのだ。解像度的には問題があるけど，撮影はいちばん簡単に済む。

オーサリングして鑑賞する

撮影が終わったら，「ステッチソフト」を使って，十何枚の写真をつなぎあわせることになる。このとき，優れたソフトとてきとーなソフトではできあがりに大きな差が出るので注意しよう。優れたソフトだと多少元の素材がずれていてもうまくごまかしてくれるが，てきとーなソフトだとごまかしてくれない。

では3つのサンプルをお見せする。CD-ROMに収録した写真の一部をアップルのQTVRASとライブピクチャ社のPhotoVistaと，PictureWorks社のSpinPanorama 2.0でつないでみたものだ。特に差が出る「重なり合った部分の処理」に注目したい。SpinPanoramaは「あまり考えてない」のがわかる（図7-01～03）。

ただ重ねるだけではだめで，レンズによる歪みを計算して滑らかにつなぐのが「優れたソフト」なのだ。QTVRASとPhotoVistaはあらかじめ使用したレンズの画角を設定するようになっているし，実際に撮影した画像を見ながら最適のパラメータを探し出せるようになっている。それによって元の画像がどれだけ歪んでいるかを判断し，滑らかにつなごうとしてくれるのである。SpinPanoramaはレンズの歪みを考慮しないため，歪みが大きな広角系のレンズを使うとうまくステッチできなくなるのだ。

階段部分を拡大すると違いがロコツにわかる（図8-01～03）。

低価格なのでSpinPanoramaを使っている人も多いと思うが，実はこれだけクオリティの差が出るということを念頭に置いておこう。

この図だと，QTVRASよりPhotoVistaのほうが優れているようだが，別の箇所（木のアップ）を見てみると，今度はPhotoVistaのほうが破綻している。ステッチのアルゴリズムによって得手不得手が出てきてしまうようだ。これも結構面白い（図9-01～03）。

こうして360度分をステッチしたら，QuickTimeVR形式で保存し，QuickTime 3.0のMovie

図7-01　QTVRASで神社の階段をステッチしたもの。素材はCD-ROMに収録されている

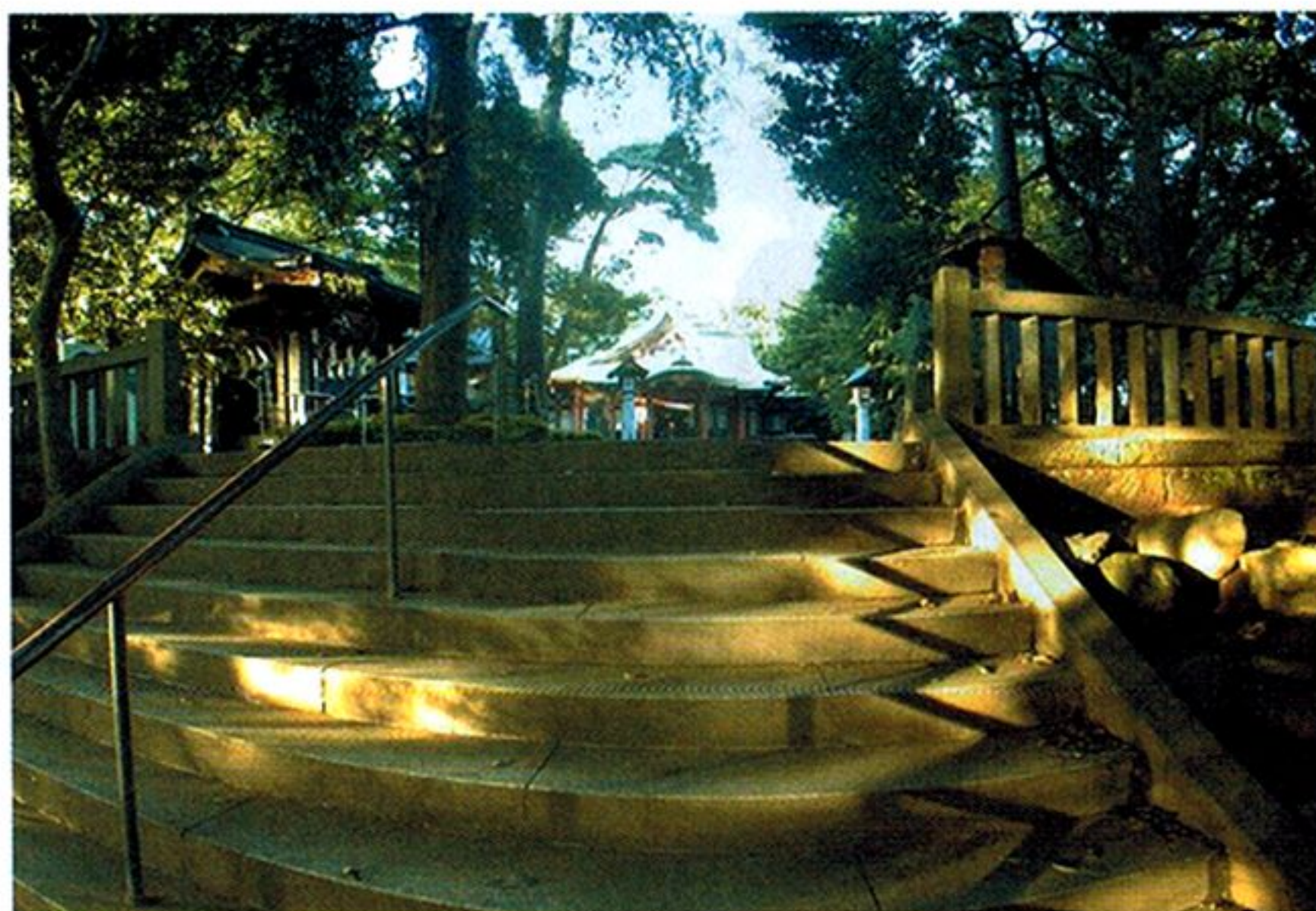



図7-02　PhotoVistaで神社の階段をステッチしたもの

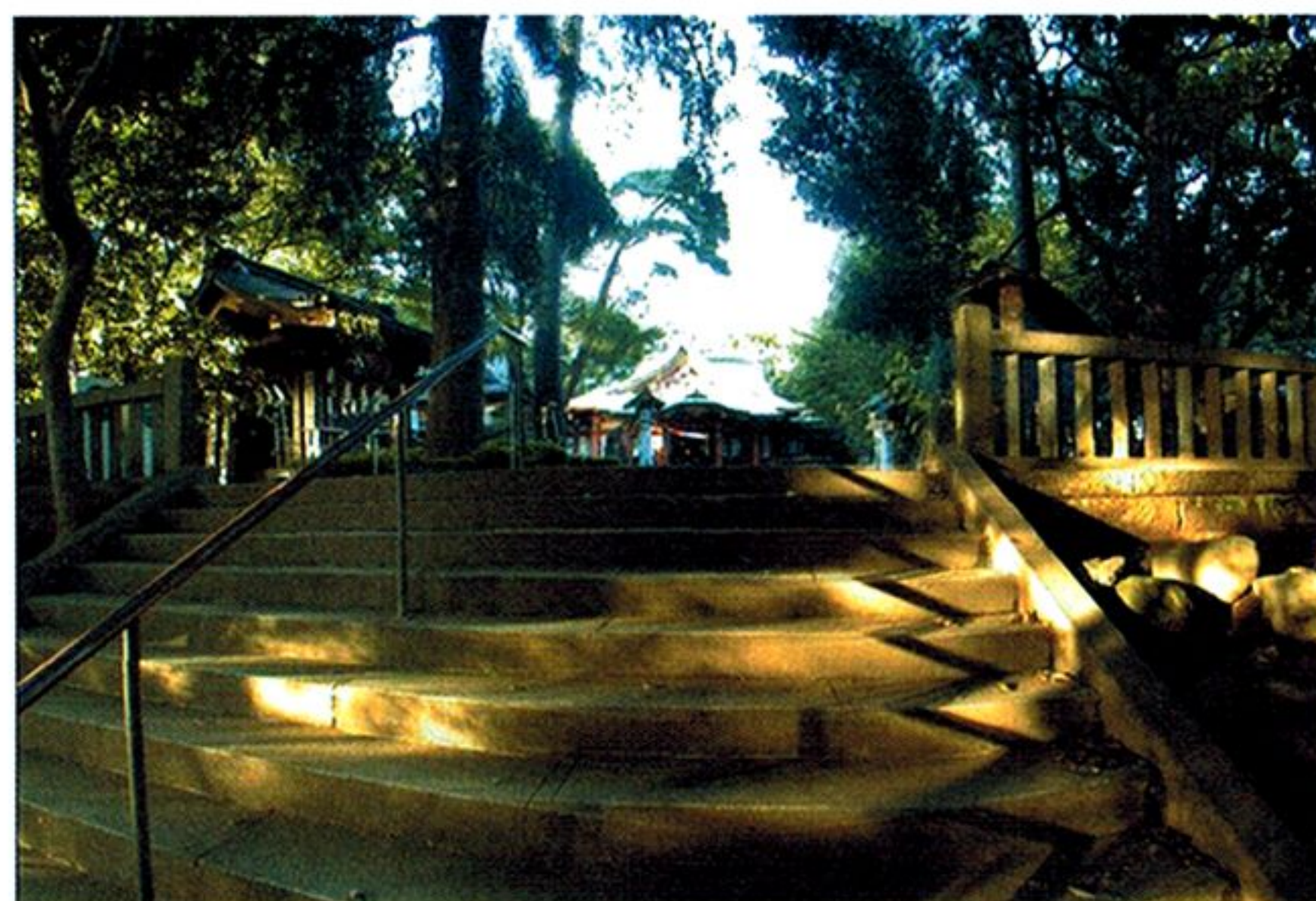



図7-03　SpinPanoramaで神社の階段をステッチしたもの

図8　階段部分を拡大した。QTVRASはこういう直線的な模様が苦手で，階段の一部をつなぎそこねている
SpinPanoramaは論外。レンズにあわせた歪みを処理していないので，レンズの端のほうはこのようにただぼかしてくっつけただけになっている

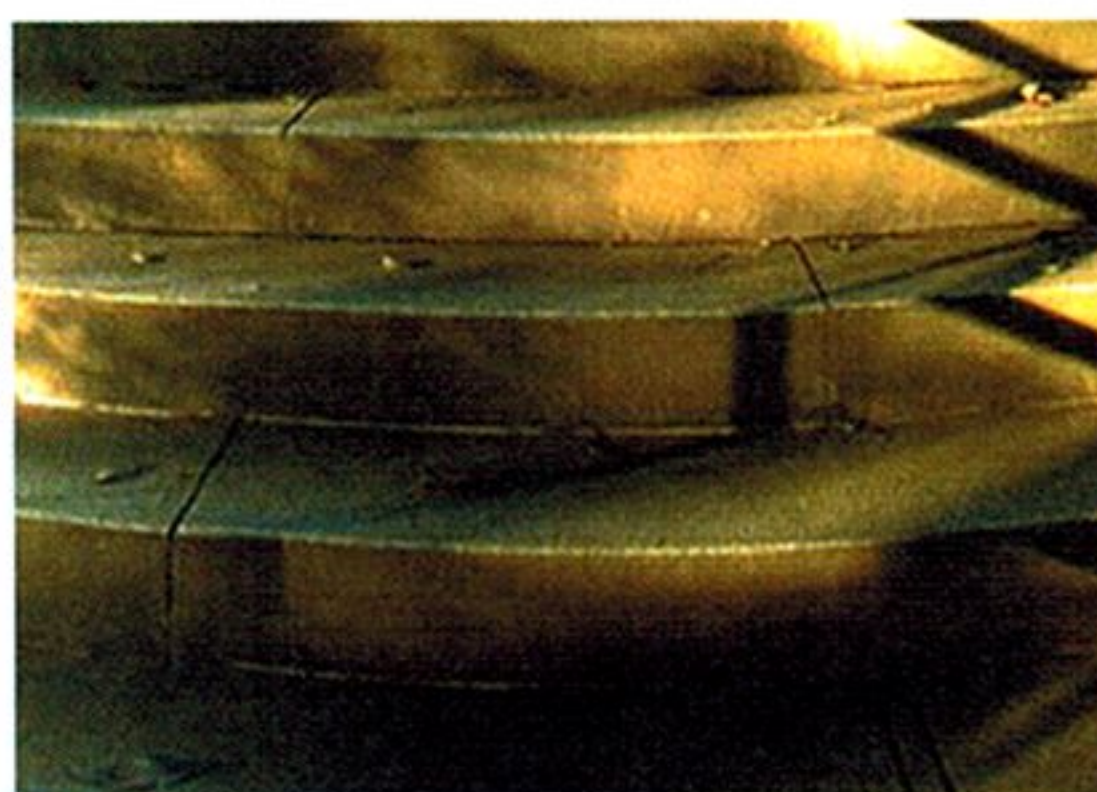



図8-01

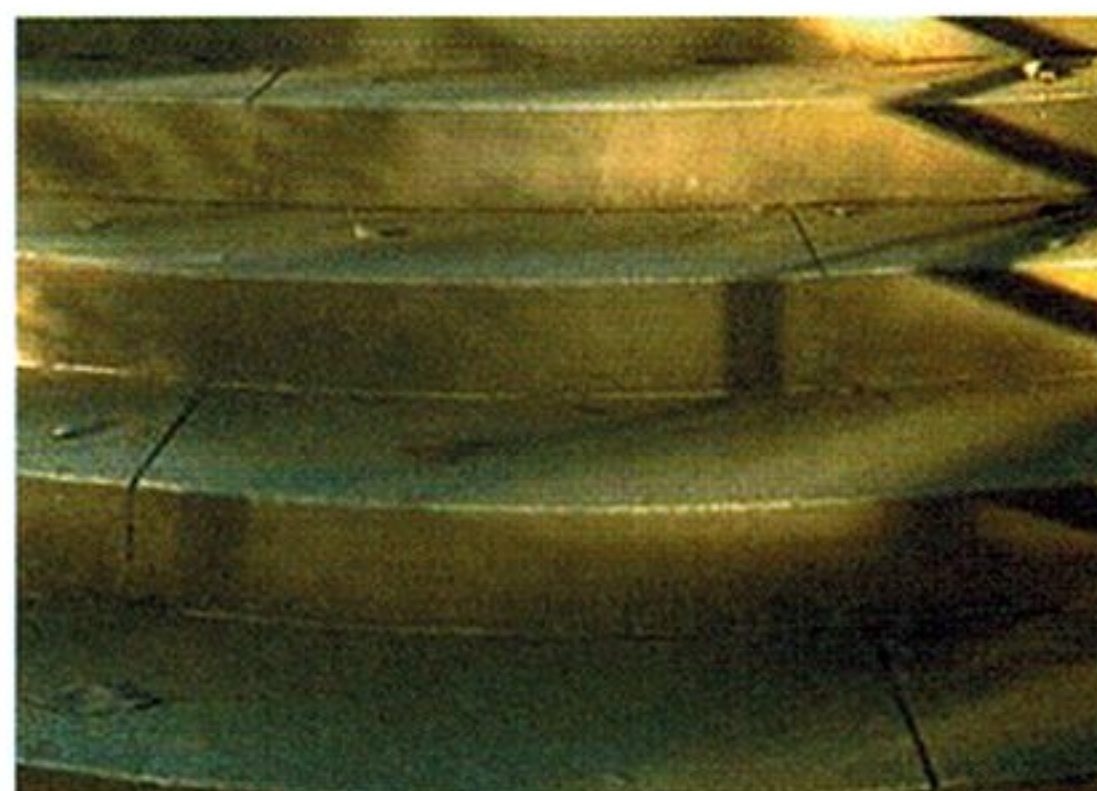



図8-02

図8-03

Playerで鑑賞する。QuickTime 3.0 for Windowsを使えば(.movファイルの関連づけがちゃんとMoviePlayerになっていれば)，問題なく見ることができる。

実際には，QuickTimeVR形式といっても，CODECをどうするか(おすすめはJPEGだ。ファイルサイズが抑えられる割にクオリティは高い)，などによってクオリティとファイルサイズに大きな違いが生じる。この中で出力を細かくコントロールできるのはQTVRASである。Windows用も探せばあると思うが，見つけることができなかった。すまん。QuickTime VR形式にしなくてもいいなら，PhotoVistaが安い割に(素材がちゃんとしていれば)クオリティの高いステッチを行ってくれる。

実際にQTVRASで作成したのがCD-ROM内の「8man-03.mov」である。カメラはCOOLPIX 900＋純正のワイドコンバータ。三脚はマンフロットのVR作成用雲台＋マンフロットの三脚。ホワイトバランスは太陽光，フォーカスは遠景固定，露出はオートだ。

撮影時の解像度はVGA。何百万画素あっても，VGAモードで撮影すれば十分なのである。なんとなれば，それでも360度分つなぐと横方向が3,000ドットを超えるでかいサイズになり，扱いづらくなるだけだからね。

QuickTimeVRのお仕事

一見，360度パノラマの鑑賞というと，360度のパノラマ画像をただウィンドウ内で上下左右にスライドしてそれっぽく見せているだけ，と思われがちだが，そんなことはない。一応「VR」と名がついているだけあって，それなりの処理を行っているのだ。

広角レンズで写真を撮ったときの話を思い出してみよう。レンズの端にある物体は歪んで写ってしまう。その歪みを補正して表示してくれるのがポイントだ。

歪ませるという処理を行ったときと行わなかったときの違いは図のようになる(図10-01，02)。

一目瞭然。ただ「左右へスライドする」だけの場合は，オーサリングした画像がそのまま表示されているわけだが，正しく歪ませたものは，ちゃんとそのシーンを正面から見たらどうなるかを計算して歪みを補正して表示している。すると，マウスのドラッグによって絵を左右に動かした場合，風景が回転しているように見えるのだ。8man-03.movをぐるぐると派手に回してみると実感できるだろう。

なお，QuickTimeVRの見方は，説明しなくても触っているとわかると思うけど，マウスのドラッグで上下左右への視点変更。左右に大きくドラッグしてそのまま保持してると高速でぐるぐる

図9-01　QTVRAS

図9-02　PhotoVista

図9-03　SpinPanorama

図9-01，02，03　階段の上に育っている木の部分。ちょうど写真と写真のつなぎ目にあたる。PhotoVistaはこういうのが苦手らしい

回って気持ち悪くなるので楽しい。＋と－のアイコン，あるいはcontrolキーとoption（Alt）キーで拡大･縮小となっている。

QuickTimeVRの実践

てなわけで，VAIO C1やらQVシリーズやらで結構軽い調子で扱われている「パノラマ」ってやつだが，実は奥が深くて，ちゃんとやろうとするとそれなりに大変なのである，って話を延々とさせてもらった。

でも，ただ単に風景をぐるぐる回したり拡大縮小して楽しんでも，作るのが大変な割にすぐ飽きちゃう。

そこで，QuickTimeVRが持っている，VRムービー中に「ホットスポット」をセットし，リンクを張るという機能や，ひとつのファイルに複数のノードを統合できるマルチノード機能を使ってみた。

たとえば，マルチノードムービー。CD-ROMに収録した8man-multi.movがそうだ。これは4つのVRを互いにリンクさせてひとつにまとめたもの。ホットスポットをクリックすることで別のムービーへジャンプできる。

QuickTimeVRにはこうしたパノラマのほかに，オブジェクトVRという形式もある。これは複数の静止画をマトリクス状に並べてドラッグでいったりきたりできるようにしたもので，回転する物体の写真を用意すればドラッグでそれを回転させられるし，それを連続して再生させれば動画として扱うこともできる。

http://www.asahi-net.or.jp/˜ax1k-ogkb/vr08.html

はその応用例である。600KBくらいあるけれども，2つのパノラマVRと複数のオブジェクトVRを組み合わせたものだ。1年以上前に作ったもので，なおかつ，このときは「手持ちカメラ」で撮影したので，クオリティ的にはかなり無理をしている。ご了承あれ。具体的には，Photoshopで補正しまくったってことなんだけどね。

urlへのリンクを張ることもできる。

http://www.asahi-net.or.jp/˜ax1k-ogkb/vrmenu/index.html

はそれを利用して目次を作ってみた。回転目次である。ぐるぐる回してると気持ち悪くなってくるのが面白い。つまり写真を使わなくても，360度分の絵があればなんでもVR化できるわけだ。

そのほか，

http://www.asahi-net.or.jp/˜ax1k-ogkb/vr00.html

にはいろんなVRを取り揃えてあるので興味のある方はご覧くださいませ。

ん？　VAIO C1でやりたい？

あれを三脚につけるわけにはいかないからなあ。でもまあ，室内なら机の上なんかにおいて，できるだけカメラを中心に回転させるようにすれば結構いけるんじゃないかと思う。そもそも画質が低いから，その分細かいズレも目立たないしね。

180度くらいならてきとーに撮ってもそれなりに楽しめるもんだし。でも360度やりたいなら，それなりに苦労したほうがクオリティも上がって楽しめるよ，と，そういうことなのだ。

それ以上に細かいノウハウを知りたいなら，編集部に要望を出してみてくださいませな。

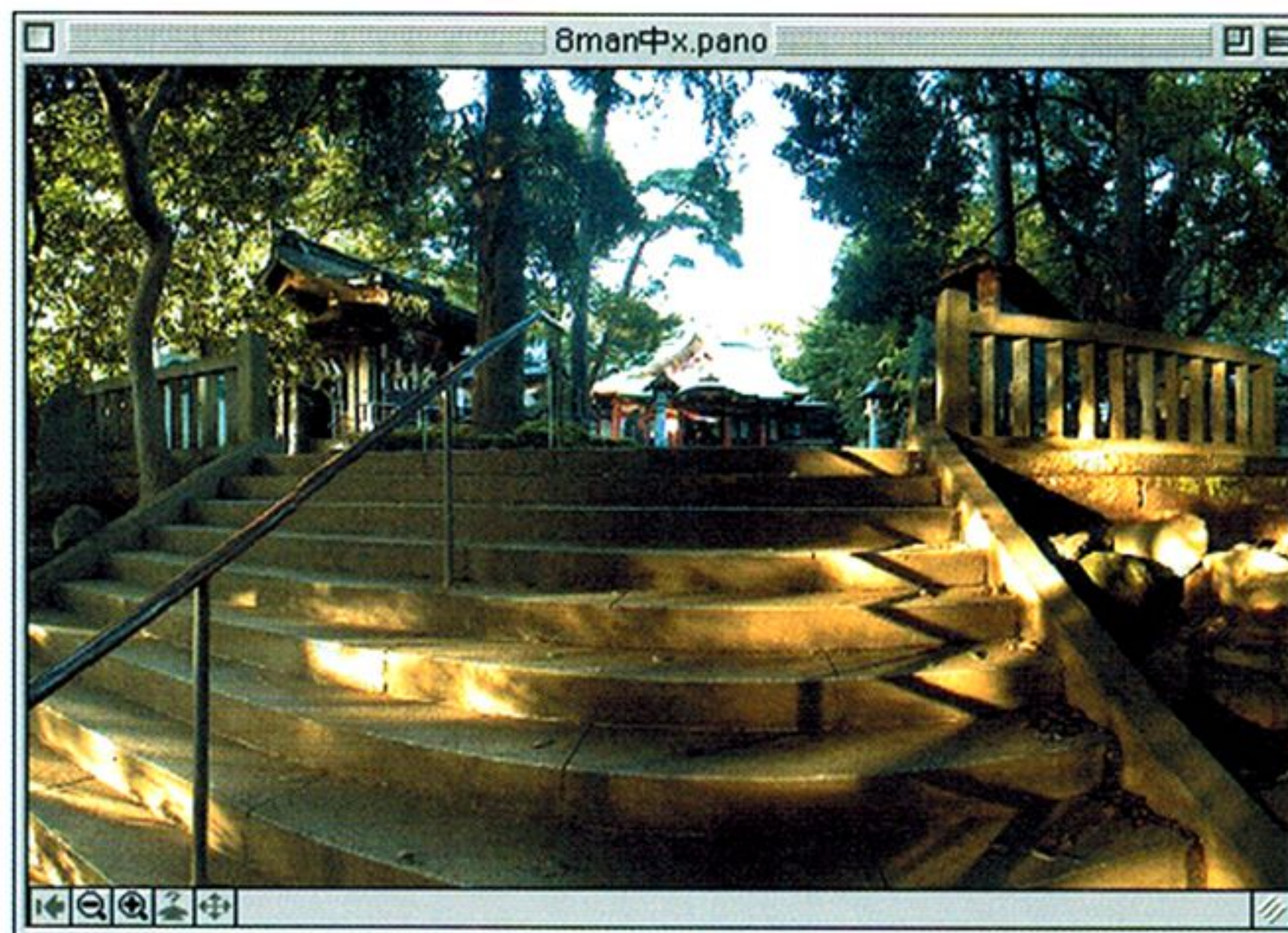



図10-01　CD-ROMに収録してある8man-03.movから。ちゃんとまっすぐに見えるよう，リアルタイムで補正している

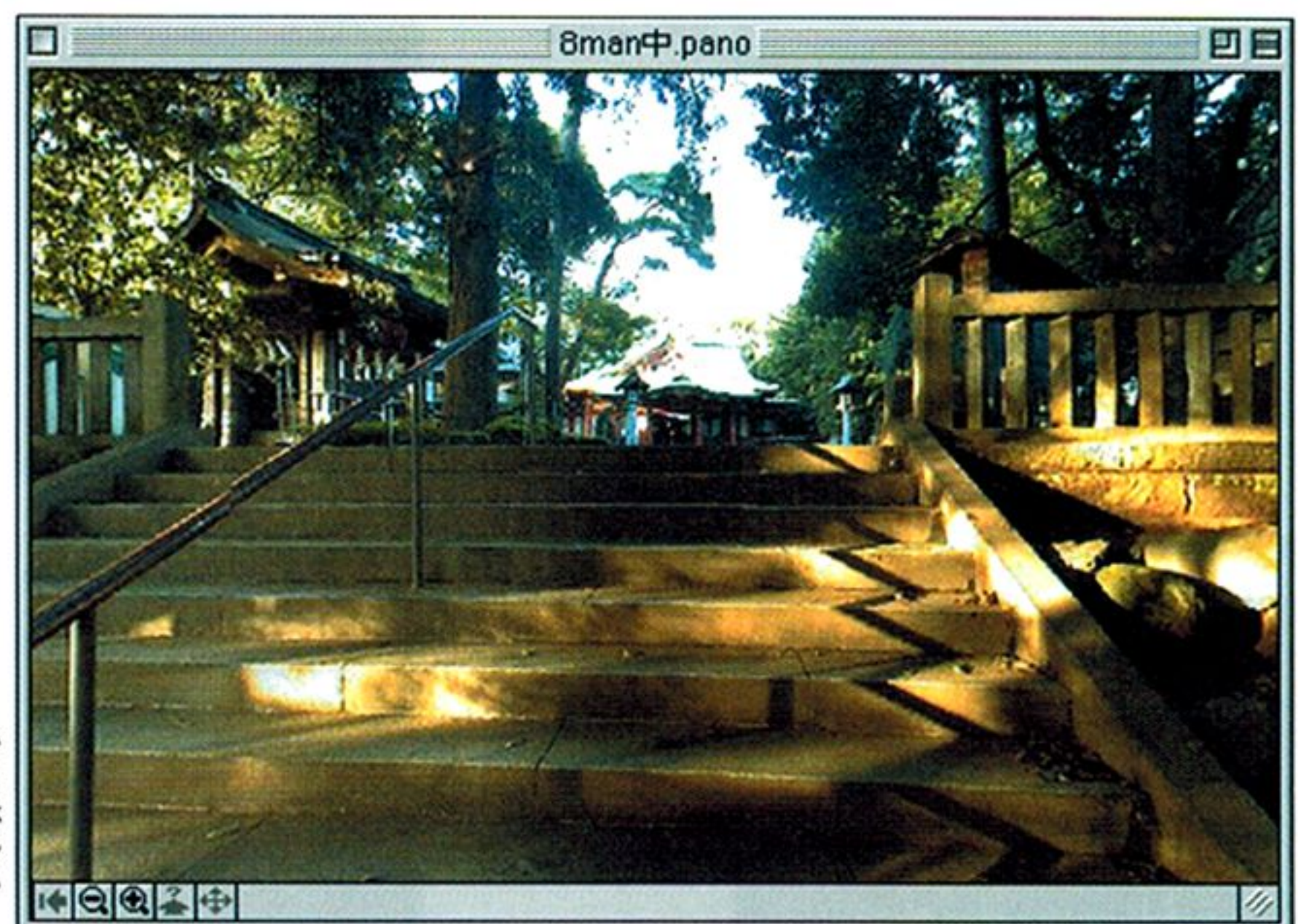



図10-02　なんの補正もしない設定でオーサリングすると，このように元の画像をただ表示するだけで，左右にスクロールさせても「絵がスライドしているだけ」にしか見えない

Oh!X 68000

いま，一からX680x0シリーズを買うとしたら，どうすればいいのかという点から，
当面する環境構築のポイントに絞って解説してみます。
でも，現有ユーザーにとっても環境をグレードアップしたいときに役立つ情報があるはず。
サラっと書いてあるので見逃さないように。

X680x0再入門

――中古で変わるか，君のコンピューティング――

電脳倶楽部編集部
あいはらてつや　Aihara Tetsuya
aihara@mankai.co.jp

CD-ROMのこと(満開青年の主張)

まず，ちょっと宣伝(というわけでもないが)。現在満開製作所では，月刊電脳倶楽部本誌のCD-ROM化への作業を検討・推進しています。また，これまで発行されたFDの電脳倶楽部をすべてCD-ROMに焼いた，いわゆる「保存版」の発売を日程に挙げています。さまざまなオトナの事情というヤツで，1～50号で1枚(予価12,800円(税込み))，51～100号までで1枚(予価14,800円(税込み))という風に分割されますが，あとは順次単年度版という形で提供していく予定です。

さしあたって「保存版1～50(仮称)」は4月末の発売を予定しています。フェスタ68の会場では製品版を手に取って見ることができるでしょう。

さて，これまで満開製作所では，都合9枚のCD-ROMを発行してきましたが，今年中にソフトウェア製品はすべてCD-ROMへと移行するつもりです。

質のいいFDの安定確保の問題とか，デュプリ発注環境の悪化とか，諸々の懸念はありますが，やろうと思えば，あと2年程度はFDのままでも物理的な発行環境(満開外の環境)を維持できないことはありません。

しかし，あえてここはCD-ROMにこだわります。なぜか。零式への展開を考えたとき，ISO9660のCD-ROMだけが唯一安価に丸ごとX68000の資産をサルベージすることが可能だからです。個人的には，LANやリムーバブルな光メディアといった選択肢をとることも可能でしょう。しかし，データ転送やハード的な接続環境のことを考えると誰もが一斉に簡単に移行，というわけにはいかないのが現実でしょう。

CD-ROMならドライブのトレイに載せるだけでどこにでも持っていくことができますし，EX68のようなエミュレータ環境にもポンと持っていくことができます。Macのネットワークドライブ上に置いておくことだってできるわけです。この手軽さは捨て難いものがあります。

なにをいまさら強調することもあるまい，という向きもあるでしょうが，いまでも「CD-ROMは使い道がなくてゴミになる」という電クラユーザーもいます。CD-ROMはメディアの保管が楽です。8ビットの頃ならカビが生えたり，伸びて読めなくなるメディアしか我々にはなかったわけです。そうして過去を捨ててきたわけです。テープや，5インチ2Dディスクやソノシートなんかを，いま読み出そうと思ったら，それこそしんどい思いをした挙句あきらめるのが関の山です。ところがCD-ROMはもっと息長く，後々に参照可能な良質なメディアだということを忘れてはいけないでしょう。

というわけで，今回は，改めてまっとうなX68000のSCSI環境を整えてもらうため，極最短コースの解説を試みています。じっくりお読みください。

激電6号表紙

激電6ゲーム
POCKET SHOT

激電6ゲーム
ACT2

激電6ゲーム
KNIGHT

では本題です

大学受験も終わって春を迎えたA君は秋葉原をうろうろしていると，偶然X680x0シリーズを扱っている中古ショップを見つけてしまいました。最近できたばかりのお店のようで，なにか惹かれるモノがあり，ついつい値段をチェックしてしまったのです。中学生の頃，ほしくても高かったソレが，そこそこ手に届きそうな価格で……。

A君はなにを思ったのか，
「Oh!Xも復刊したことだし，X680x0ユーザーになろう」
と，決意してしまいました。

本体入手のポイント

そんなわけでさっそく情報収集を始めます。最近は中古といっても某有名店では高値で安定してしまったり個人情報誌に「売ります20万円」(おいおい)なんて載ってたりしますが，WebサイトやNIFTYの中古情報，満開ネットなど，ついている価格が高いか安いか理解しているX68000ユーザーがいるところで探すのもいいでしょう。

A君も家にはAT互換機があり通信環境も完備していますのでいろいろと探して回ってみました。

まず，買う前にチェックしなければいけないのは2Mバイト以上メインメモリが搭載されているか，SCSIが標準でついているか，または10MHz機の場合はSCSIボード付きのものを買えるかどうかという，メモリとSCSIの2点です。もちろん完動品でないとあとで困りますが，この際，改造の跡があるとか，側板がないとか，汚れなどは「些細なことなので気にしない」ことにしましょう。

1Mバイトしか積んでいない機種では，専用の1Mバイト増設メモリの入手が非常に困難なので，必ずメインメモリが2Mバイトに増設済みであるか購入前に確認しましょう。

さらにメモリ増設する際にも，本体背面の拡張スロット用増設メモリボードは既に本体に2Mバイト搭載していることを前提に作られているので，「本体のメインメモリが2Mバイト以上」であることは必須です。

それから周辺機器を接続しようにも，SASIのHDDなんてジャンク品でもそうそうお目にかかれませんので，SCSIをなんとかしてつけなくてはなりません。

標準搭載機ではSUPER，XVI，Compact XVI，X68030(CZ-500C)，X68030(CZ-300C)が該当します。SCSIに関しては，サードパーティ製(ツクモ，サコム)SCSIボードでも特に問題ありません。また満開製作所のMach-2(販売終了)はいまでもサポートが受けられます(ただし，ROMのバージョン情報などを別途チェックしなければなりません。またPROシリーズではMach-2は使えません)。

ツクモのTS6BS1 mkⅡ(ダッシュを除く)は若干問題があるので避けたほうがよいでしょう。ソニーのNEWS用の5インチMOとハドソン製SCSIボードのような，レアアイテムに近いセットも昔はありましたが，そういうものでも問題はありません。ジャンクでもいま持ってたらちょっと自慢できるかもしれませんね。

Mach2pが今年中に出てくるので，ボード増設は，PROユーザーを含めてまだなんとかなるでしょう。

最近はコンデンサが死んだ電源ユニットのせいで動作が不安定になったり，動かなくなったり，という本体も多々見られるようになりましたが，基板に損傷がなければ電源部さえ取り替えれば動くようになることがあります。非常にもったいないので現有ユーザーも本体の動作が不安定ならATX電源接続キットを試してみる価値はあるでしょう。

お金がなく，簡単な電気工作の心得がある人は，安いジャンク品の本体と，満開のATX電源接続キット(ATX用の電源ユニットを使うための基板キットです)を購入してみる，というのもX68000ユーザーになる最終手段としてアリかな，と思われます。

さてさて，A君はネットの知人のつてで，めでたくXVIを4Mバイト搭載した状態で200Mバイトの外付けHDD付きという良品を手に入れることができました。でもマニュアルがなく，付属のシステムディスクのマスターも読めなくなっていました。もちろんバックアップディスクをもらいましたので，とりあえず，XVIに付属のHuman68k ver2.03とSX-WINDOW ver1.1の環境はひととおり手に入りました。

システムディスクの入手

シャープが取り扱いをやめてしまったため，純正システムキットの新品での入手は100%不可能です。ちなみ最終バージョンは「SX-WINDOW ver3.10システムキット」で，これに最終版のHuman68k ver3.02のシステムディスクが含まれています。できれば，彼はこのセットを使いたい，と考えました。

方法としては，

- ネットや雑誌で「譲ってください」告知を出してみる
- 電脳倶楽部で「ください」告知を出してみる
- そのほか告知を出してみる

といったことが考えられます。マニュアルも読みたいので，セットそのものを譲ってもらうのがベストですが，当面は我慢ということにしました。

ディスクだけなら，周りの人から(ピー)してもらうというダークサイドな手段がないでもないですが，マニュアルがやっぱり重要ですね。

マニュアル以外でも，X68000の解説本は何点か出版されています。中古書籍を探してみてもいいでしょう。記述は古くなっていますが，おおざっぱに概要をつかむことができます。

SRAMのチェック

使われていた環境がまったくわからないマシンを入手した場合や，最初から環境を構築するつもりなら，まず最初にSRAMをクリアしておいたほうがよいでしょう。SRAM常駐型のウィルスが入ったまま中古市場に出てしまうというケースがあるからです。XVI以降のマシンならば，CLRキーを押しながら電源を投入することで，SRAMクリアのメニューが出てきます。これだと簡単に完全にクリアすることができます。

10MHz機の場合，別途このようなクリア用のプログラムを用意する必要があります。

```
*-- アセンブラです
*-- ｽｰﾊﾟｰﾊﾞｲｻﾞ
        clr.l       -(sp)
        .dc.w       $ff20
*-- SRAM Write ON
        move.b      #$31,$E8E00D
*-- SRAM ｼｷﾍﾞﾂｼ CLR
        clr.l       $ED0000
*-- ﾘｾｯﾄ
        moveq       #$00,D0
        trap        #$0A
* メモリサイズも初期化されるのでFD に
* SWITCH.X の入った起動ディスクを用意
* するのを忘れないように
```

ただSX-WINDOWをメインの環境にして使っている人から譲ってもらった場合は，SX-WINDOWで使用する情報がSRAMに書き込まれているので環境を改めて作り直すのでない限り消去しないでよいでしょう。

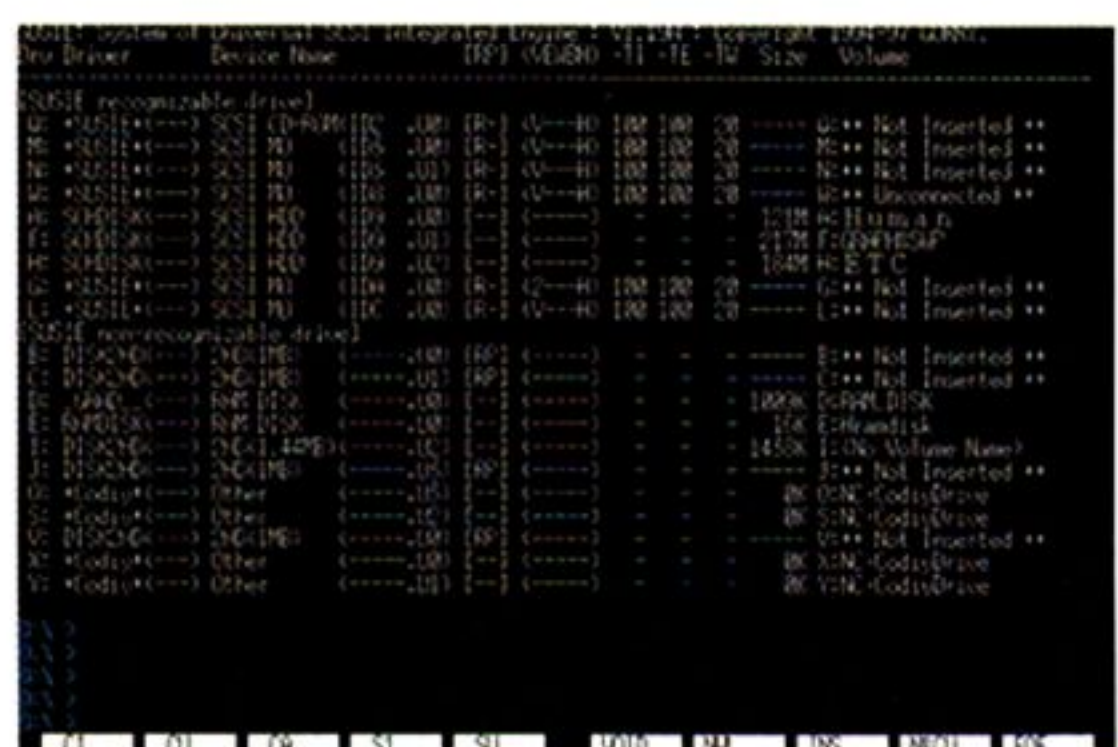
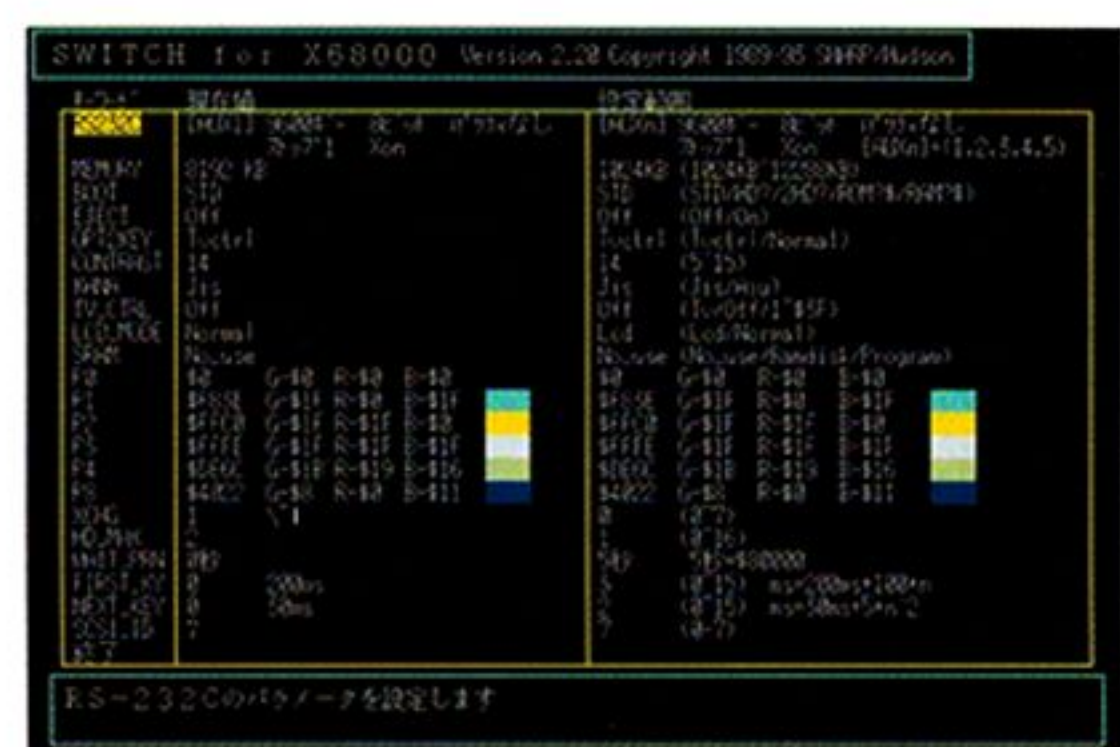
また，いままで問題なく使われてきたことがわかっているなら，SRAMクリアはスキップしてもかまいません。

外部記憶媒体を入手しよう

A君の入手した本体は，200MバイトのHDD付きとはいえ，相当古くなっています。クラッシュにおびえて暮らすわけにもいきません。まずバックアップ手段を考えましょう。それから，HDDの増設もしたい，いまや必須のCD-ROMドライブも接続したいと考えています。

HDDのポイント

SCSIのHDDは，下位互換性があるのでFast SCSI，UltraSCSIの製品がX68000でも使用可能です。ただし「WideSCSI」ではバスとコネクタ

FIM.X画面

SUSIE.X画面

SWITCH.X画面

形状が違うので変換アダプタが必要になります。WideSCSIの製品はほかよりもともと高いので，FastSCSIや，UltraSCSIの安価な製品で十分です（最近のUltraクラスのHDDはWideのほうが安いのですが）。

MOドライブのポイント

3.5インチMOドライブは230～650Mバイトの製品が多いですが，価格と手軽さでは230Mタイプのもので十分でしょう。ただし，一時期のオリンパスドライブのように，X68000でまれに使えないドライブも存在したことがあります。ほとんどのドライブは大丈夫です。不安な場合，別々のメーカーのドライブをひとつずつ買って，ひとつはAT/Mac用，とすることもできます。が，パソコン通信などでほかのユーザーの使っているドライブの情報をチェックしておけば困ったことにはならないでしょう。月刊電脳倶楽部にも使えるデバイスを紹介するコーナーが不定期であありますが，製品サイクルが早いので接続する際の参考程度と考えましょう（それなりに接続に役立つ情報もありますが）。オーバーライト（OW）タイプのドライブは，確かに書き込みもOWタイプでないドライブより速いですが，メディアも高価で専用になります。他人とのやり取りを考慮して，通常のものにしたほうがいいでしょう。あとは，バックアップ用という割り切りも時には必要です。

CD-ROMドライブのポイント

CD-ROMドライブは，40倍とか32倍速などの高速なものでも接続可能ですが，そんなパフォーマンスは期待できません。SCSI2のものであれば（SCSI3可。SCSI-1ではそもそもCD-ROMに対応していない），特に接続に困るドライブはありませんので店頭で安いものを選べばよいでしょう。これも製品サイクルが非常に短く，24倍速ドライブが出た頃は，満開のスタッフも喜んで買ってきてはチェックして記事にしていましたが，最近はそうした熱も一段落してしまっています。最近のドライブはほとんどSONY系コマンド，またはTOSHIBA系のコマンドでCDDAトラックの読み出しができますので（CD2PCMなど），選ぶのには苦労しなくてすみます。ただし，CD-Rドライブはcd-ROMとしてのデバイス番号を返しませんので注意が必要です。

ZIP，JAZドライブなども接続可能ですが，これらについてはまた日を改めましょう。

SCSIボックス

外付けドライブをメーカー品の筐体ごとに買ってくると配線と設置場所に非常に困ります。4段くらいのSCSIボックス（電源付き）を買ってきて，ベアドライブ（筐体なしのドライブむき出しのもの）を買い足して増設していくのもよいでしょう。なかには電源の弱い2段くらいのものも売られていますが，電源が弱いものはドライブの動作が非常に不安定になりますので避けます。

COMMAND.X

なにも知らない人のために念のため説明しますが，OSたるHuman68kを起動させると，通常の環境ではCOMMAND.Xというシェルが起動します。これで実際にX68000に命令を下せる状態になります。つまりこの上で，いろいろなコマンド（プログラム）が実行できます。コンパイラもエディタも，環境設定のためのコマンドも，もちろんゲームや電脳倶楽部なども，シェルの上で実行されます。X-BASICなどもシェルから起動する外部コマンドの一種というわけです（編注：外部プログラム自体をシェルとして動作させることもできますが，一般的には，外部プログラムを起動することを主目的としたプログラムのことをシェルと呼ぶわけです）。

それから，COMMAND.Xの上では，バッチファイルが実行できます。COMMAND.Xがテキストで書かれたバッチスクリプトの内容を解釈して，外部コマンドを連続で実行したり定型の処理をしたりします。

いよいよセットアップ

OPT.1キーを押しながら，本体がHuman68kのシステムディスク（FD）で起動できたのを確認したら各種デバイスの初期化をしなければなりません。

っと，その前に。X68kではブート（起動）情報をSRAMが管理しています。OPT.1キーを押しながら起動すると，強制的にSTDモード（FD0から順に見てみつかったデバイスから）起動するようになりますが，起動デバイスには，FD0～3，SCSI0～7，ROM，RAMなどのモードが存在します（10MHzでSCSIのない機種では，SCSI0～7のところが，HD0～15になります）。

SCSIのHDD環境では，起動HDDのSCSI-IDに設定しておくのが一般的であり，これはSWITCHコマンドという専用の外部コマンドで設定します。

とりあえず，FDからの起動ができたら，システムディスクにあるSWITCH.Xを実行して，自分のマシンがどのような設定になっているのか確認しましょう。メニュー形式なので，難しくないと思いますが，HDDに新しい環境を作るまでは，STDに設定しておきましょう。

Human68kでは，起動システムのあるドライブからドライブ名を割り当てていくようになっています。FDで起動した場合は，FD0，1がA:，B:となりますが，HDD（仮に3ドライブ）から起動すると，HDDのほうからA:，B:，C:，そのあとFDにD:，E:が割り当てられます。このあたりは，AT互換機などと違うので気をつけましょう。

環境の書き換え

X68000にはそこそこ強力なED.Xというエディタが標準添付されています（SX-WINDOW3.0以降ではシャーペンという，これも強力なエディタが付属しています）。必要最低限のコマンドですが，インライン入力もできますし，割とまともなテキスト環境を持っています。これで，動作環境やドライバの設定をするためにCONFIG.SYSをカスタマイズしていくわけです。

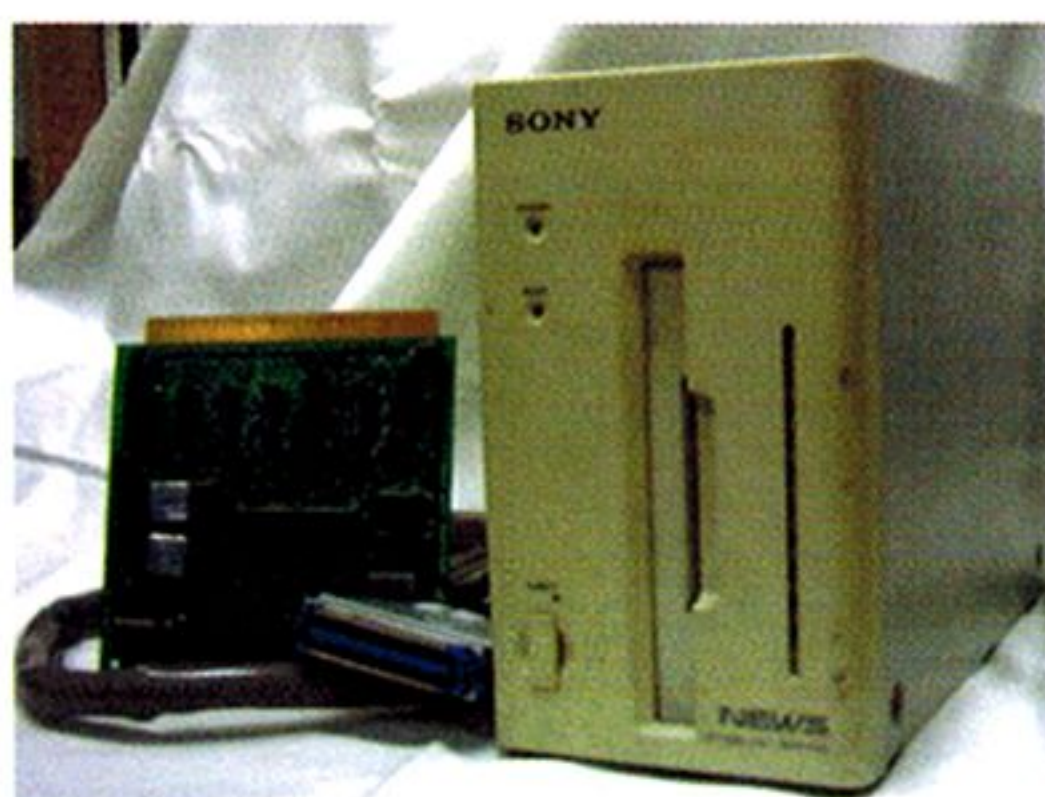
```
ED CONFIG.SYS  <CR>
          <CR>はENTER
```

としてもらえばHuman68k（ver.3）の標準環境では，

```
FILES      = 30
BUFFERS    = 20 1024
LASTDRIVE  = Z:
KEY        = \KEY.SYS
USKCG      = \USKCG.SYS
BELL       = \BEEP.SYS
DEVICE     = \SYS\PRNDRV.SYS
DEVICE     = \SYS\RSDRV.SYS
DEVICE     = \SYS\FLOAT2.X
DEVICE     = \SYS\ASK68K.SYS /DB:\X68K.DIC
 /E\ASK\ENV1.ASK
DEVICE     = \SYS\OPMDRV3.X #180 /P64 /OPM
DEVICE     = \SYS\FDDEVICE.X
DEVICE     = \SYS\HISTORY.X /D\HIS\ /SH2,8,4
ENVSET     = 512 \STARTUP.ENV
EXCONFIG   = \SYS\CONFIGED.X
```

と表示されます。これを，

```
FILES      = 30
BUFFERS    = 20 1024
LASTDRIVE  = Z:
KEY        = \KEY.SYS
USKCG      = \USKCG.SYS
BELL       = \BEEP.SYS
#DEVICE     = \SYS\PRNDRV.SYS
#DEVICE     = \SYS\RSDRV.SYS
```

シャープマニュアル

ソニーMO

SCSIボックス

```
    DEVICE     = \SYS\FLOAT2.X
    DEVICE     = \SYS\ASK68K.SYS /DB:\X68K.DIC
/E\ASK\ENV1.ASK
    #DEVICE     = \SYS\OPMDRV3.X #180 /P64 /OPM
    #DEVICE     = \SYS\FDDEVICE.X
    #DEVICE     = \SYS\HISTORY.X /D\HIS\ /SH2,8,4
    #ENVSET     = 512 \STARTUP.ENV
    #EXCONFIG   = \SYS\CONFIGED.X
```

のように変更してみます。

登録されているところを消すのが嫌ならば，上記のように，行頭に#や!を書いておけば，登録されなくなります。浮動小数点演算ドライバ(FLOAT2.X)と日本語FEP(ASK68K.SYS)を登録しておけばよいです。上記で#で消したものは，当面なくても環境構築に支障がないということです。便利な機能もありますが，それはおいおいマニュアルとにらめっこしてください。特に，OPMDRV3.Xのようなドライバは，用途によって複数のフリーの音源ドライバに置き換えて使い分けるのが普通なので，CONFIG.SYSでは組み込まず，起動後に登録するようにしましょう。

Human68k ver.2.0の SCSIデバイスドライバ

Human68k ver.2.0では，SCSIDRV.SYSも登録しておくことが必要です(フリーの代替ドライバもありますが，場合分けなどが煩雑になるので，とりあえずこの記事では取り上げません)。

```
    DEVICE = \SYS\SCSIDRV.SYS /ID0
```

複数のドライブを持っている場合は，そのドライブ分，ID番号ごとに登録します。

ちなみに，ver3.0以降を使えばCONFIG.SYSになにも記述しなければ自動的にSCSIのドライブを検索して登録してくれますのでこの作業は必要ありません(ただしその分起動が遅くなります)。

一応，A君はATユーザー(もしくはMacユーザーでも可)で，ここまでは日本語の入力やテキストエディタを使ったことがある，ことを前提にしてきましたが，以降はさらに，LHAを使ったことがあるという前提で話を進めます。

HDDのセットアップ

まずは買ってきたHDDのフォーマットから。Human68k 2.0のFORMAT.XではXelent30など68030以上のMPUを搭載していると問題がありますので該当機種ではMPUキャッシュをオフにしたうえでフォーマットします。手順としては，Human68k ver3.0xでも2.0xでも変わらないのですが，FORMAT.Xをメニューモードで起動し（単にFORMAT <CR>で起動するだけですが）物理フォーマット「装置初期化」を選択します。時間がかかるのでしばらくはコーヒータイムになります。

それが終わったら論理フォーマットで「領域確保」します。領域(パーティション)はドライブ名に使えるのが26文字ですから，あまり細かく分けず，500Mバイトくらいの単位で確保してやるといいでしょう。このパーティションひとつが1ドライブとして扱われます。この際，起動ドライブとして使うパーティションには「システムファイルの転送をする」設定にしておきます。データ用に使うドライブは，「転送しない」でOKです。ただし複数のシステムを分けて使いたい場合には，複数のパーティションにシステムの転送を設定します。

フォーマットが終了すると，起動するためにはリセットが必要，というメッセージが表示されますが，これはとりあえずNOにして，FORMAT.Xを終了します。

そのうえで「GOVERHD」(ジーオーバーエイチディと読みます。ギガバイトを超えるHDのための，IPLとデバイスドライバの修正ソフトです)を実行してIPLを書き換えてしまう，というのがベストなチョイスです。特に1Gバイトを超えるHDDというか，最近はみな4Gバイト以上のものが売られていますので必須です。

※GOVERHDは電脳倶楽部123号掲載

筆者の環境では，4Gバイトより大きなHDDをフォーマットしたことがないのですが，最大で2Gバイト以下のパーティションに切り分けたほうがよいと思われます。ちなみに筆者の環境では，400Mバイトが4つと700Mバイトが2つ切っています(CD-ROM制作には700Mバイト2つ程度は必要ですね)。

```
GOVERHD <id> <CR>
※<id>は該当するHDDのSCSI ID番号
```

とするだけです。

この状態では，HDDはまだドライブとして認識されていないのでいったんリセットし，FDで再起動します。そのあとで，FDがカレントの状態で，

```
copyall *.* <新>:\
copyall <旧>:\*.* <新>:\
※<新><旧>はHDDのドライブ名
```

などとして，FDや古いHDDの内容をコピーしていけばよいわけです。ドライブ名を確認するには，DRIVEコマンドが標準でついていますが，後述のSUSIE.Xをオプションスイッチをつけずに実行したほうが，より詳しい状態が表示されます。

CD-ROMのセットアップ

CD-ROMを使うには，多少必要な知識が多くなります。問題はデバイスドライバです。シャープは最後までCD-ROMのドライバを供給しませんでしたので，計測技研製か，フリーソフトが必要になるのです。古いSCSI1以前のCD-ROMドライブは，すでに新品入手はできないでしょう。というか，ドライブだけあっても，今度はドライバの入手が困難ですので，タダでもらえる話があっても避けたほうがいいでしょう。というか，避けてください。まれにあるんですが，満開製作所に電話されても，どうにもフォローしてあげられませんので。

ここでは，SCSI2のドライブの入手を前提に話を進めます。ドライバはフリーソフトのSUSIE.Xを使用します。

SUSIE.Xを¥SYSディレクトリにコピーした場合，SUSIEは，月刊電脳倶楽部111，112，113号，123号でバージョンアップされたものが載っていますので，そちらから入手可能です。

CONFIG.SYSに以下の1行を追加して起動します。

```
DEVICE = \SYS\SUSIE.X -ID6 -U1 -V1 @:
※ID番号はドライブにあわせてください
※-U -Vは，SUSIEのドキュメントを参照
```

これで，とりあえず，ISO9660のCD-ROMが読めるようになります。激光電脳倶楽部を購入すれば，GCC＋LIBC＋HAS＋HLK(すべてフリー)での開発環境や，インターネット接続，パソコン通信ソフト，グラフィックツールやNetBSDなどもカテゴリごとにひととおり網羅して揃えることが可能ですし，いろいろなCD-ROMをデータディスクとしてウニウニといじり回すこともできるようになります(編注：宣伝ぽいですが，X68000をまともに使用するには数多くのフリーソフトウェアが必要，ないし，あったほうが格段に便利です。多くはパソコン通信やインターネット経由でも入手可能ですが，激電一発のほうが手っ取り早くて面倒がありません。というか，そのためにCD-ROM接続について真っ先に解説してるわけですね)。

MOのセットアップ

とりあえず，メディアもそう高くなく持ち運びにも便利ということで，価格も手頃な230Mバイトのドライブを買ってきました。X68000ユーザーはMOの所有率も高いので，そのあたりも考慮してのことです。で，MOのフォーマットですが，ほかの機種とのデータ交換を考慮すると，FORMAT.X以外でフォーマットするのがよいでしょう。

MO用にSUSIEを登録するには，CONFIG.SYSに1行追加します。

```
DEVICE = \SYS\SUSIE.X -ID1 @:
※-IDオプションでMOドライブのID設定にあわせてください
```

市販メディアはたいてい物理フォーマット済みなのでWindowsでいう，いわゆるクイックフォーマットのようなフォーマッタを使います。これにはフリーソフトの「FIM.X」を使います。

※電脳倶楽部123号ほか

これでIBM STANDARD MOタイプのディスクを作り，SCSIデバイスドライバSUSIE.Xを登録した環境で読み書きします。

ちなみに，FORMAT.Xでもフォーマットすることはできますが，この場合はX68000でしか読むことはできません。

さて，一気にSCSIを導入するところまで書いて予定の紙幅が尽きかけ(？)たので，今回はこれにて終了，という感じですが，ほかにも周辺機器はたくさんありますし，たくさん消えています。現時点ではこの記事は有効ですが，プリンタ，スキャナなど，最近のものではコマンド表すらメーカーから出てこないので接続の厳しいものもあります。次回書く機会があれば，ほかの周辺機器や他機種とのデータのやり取りに関して書きたいところです。

Oh!X 68000 零式

前回（Oh!X 復刊第１号）では，「X680x0用ハードウェアのこれまで，現況，そして将来」と称して，各種ハードウェアを紹介しました。しかし，将来については書き切れなかった部分が多く，心残りでした。
さらに，同人ハードウェアの一部を除いて，なにか企画があると満開製作所に持ち込まれて，製品化される流れができつつあります。
そこで本稿では，零式も含め，これから満開製作所がどのような商品展開を行っていくかということについて紙幅をいただき，ご紹介していきます。
一部，他誌で報道した内容と重複している部分がありますが，業務の都合上お許し願いたいと思います。

これからのX680x0そして零式

満開製作所
中村隆生　Nakamura Takao

前稿からの変化

まず，東京システムリサーチさん（以下 TSR さん）の X680x0 関連製品が完全に販売終了となりました。これで，名実ともにX680x0のサードパーティは満開製作所だけとなってしまいました。実際に，具体的な動きがあるというのはコラムに書いているとおりです。

最後の砦ってやつですね。なんか格好いいな。ていうことをいっている場合ではなくなってきましたが。

・零式について

前号でも発表しましたとおり，満開製作所では X680x0 の後継機を企画しています。前号では「互換機」と書きましたが，正しくは「後継機」と称するべきでしょう。

オリジナルなアーキテクチャを持たせて，X680x0と直接互換する部分はなくしますので，イメージとしては68060で動くEX68のような感じになってしまいます。

「それならAT互換機でEX68を動かしたほうがいいじゃん」とおっしゃる向きもありましょうが，実機があって動いているという状態を超えるものはありません。

我々が作りたいのは仮想の環境ではなく，電気が流れる物理的なマシンなのです。我々は，計画名「零式」を開発するべく設計を行っています。なお，「零式」はあくまで計画名であり，将来変わる可能性があります。

設計のメインは，本誌でお馴染みの桑野氏にお願いしており，現在，すでにバスコントローラの設計に入っています。具体的には，68060 ボードとして設計できる寸前のところまできています。ヴァンティス・ジャパン社（というか，まさちく工房）との共同開発によるSDRAMコントローラも，おおむね完成しています。PCI バスブリッジに関しても，問題ないと断言できるところまできています。

これらの成果を踏まえ，試作機「九九式試製一型」を製作します。これが，いわゆるα版になります。このときのマザーボードは，まるで工業用のようなバックプレーンボードになる予定です。この試製一型で，68060ボードと表示周りのデバッグを行います。

表示周りの設計は流動的ですが，RIVA128が最有力候補です。流動的なほうが，将来足を引きずるような構成にならないと考えています。ものは考えようというやつですね。

さて，バスコントローラさえ変更すれば，MPUは別に68060専用というわけではありません。零式のメインバスであるMPUバス「Qバス」では，MPU依存部分を最低限にします。PowerPCやSH4などの，ほかのMPUも，バス変換さえすれば使えるような設計になっています（ただし現行Qバス自体は32ビット幅）。

68060も含めて，基本的にはバスを握りっぱなしのデバイスはないことになっています。そして，バスマスタになりたいデバイスが，バスリクエスト（BR#）を出します。バスコントローラのアービタが，これらのBR#信号を見張っており，必要に応じてバス使用許可（BG#）を出します。バスマスタは，動作している間，バスがビジーであることを示す信号（BB#）を出します。L2キャッシュが十分に大きければ，マルチMPU環境での動作効率がとてもよくなるものと思われます。

この機構に従う限り，極端にいえばバスマスタはなんでもよくなりますから，たとえばメインMPUに68060を使用し，サブにも68060を使用するといった贅沢な動作や，メインにK6-III を使用し，サブにSH4を使うというような派手な使い方も，OS さえ対応していればできることになります。

また，書き込みは連続して行われることが多いため，ポストライトバッファが用意されます。CPLD であるVF1のSRAM機能を活用して，最大32 回分の書き込み（128バイト分）のデータを蓄えることが可能です。厳密には違いますが，書き込みキャッシュのようなものだと考えてください。この機構により，Qバスの効率がかなりよくなるはずです。

以上のような機構がOSレベルでもサポートされれば，いきなりマルチMPU環境が構築できるわけです。ただし，いわゆる仮想マシンではあり

図1 九九式のメモリマップ

68060 Address

Address	Area
0000_0000	MainMemory
4000_0000	MainMemory
8000_0000	PCI
0000_0000	I/O Area
FFFF_FFFF	

I/O Area

Address	Area
C000_0000	(Reserved)
D000_0000	(Reserved)
E000_0000	Q-Bus → Q-Bus
F000_0000	SS-Bus → SS-Bus
FFFF_FFFF	

Q-Bus

Address	Area
E000_0000	Q-Bus I/O
E200_0000	Exp.Slot#1
E400_0000	Exp.Slot#2
E600_0000	(Reserved)
E800_0000	68060 Slot
EA00_0000	(Reserved)
EC00_0000	ColdFire Slot
EE00_0000	(Reserved)
EFFF_FFFF	

Q-Bus I/O

Address	Area
E000_0000	Q-Bus Device
E010_0000	(Reserved)
E020_0000	(Reserved)
E030_0000	(Reserved)
E1F0_0000	(Reserved)
E1FF_FFFF	

Q-Bus Device

Address	Area
E000_0000	Reserved
E000_2000	Cache/Mem. Controller
E000_4000	Q<->PCI Bridge
E000_6000	Q<->SS Bridge
E000_8000	Interrupt Controller
E000_A000	Reserved
E00F_E000	Reserved
E00F_FFFF	

SS-Bus

Address	Area
F000_0000	SS-Bus I/O
F200_0000	Exp.Slot#1
F400_0000	Exp.Slot#2
F600_0000	(Reserved)
F800_0000	(Reserved)
FA00_0000	(Reserved)
FC00_0000	(Reserved)
FE00_0000	IOCS-ROM
FFFF_FFFF	

SS-Bus I/O

Address	Area
F000_0000	SS-BusDevice
F010_0000	(Reserved)
F020_0000	(Reserved)
F030_0000	(Reserved)
F1F0_0000	(Reserved)
F1FF_FFFF	

SS-Bus Device

Address	Area
F000_0000	System Control
F000_2000	Super-I/O (ISA-I/O)
F000_4000	Universal I/O (Stick)
F000_6000	Universal Timer
F000_8000	(Reserved)
F00F_E000	Reserved
F00F_FFFF	

図2　九九式のブロックダイアグラム（ver.0.5）

X MHz

Q-Bus
(X MHz)

PCI-Bus
(X/2 MHz)

Main MPU
68060RC50

Buffer

Q<->PCI
Bridge
(FPGA)

Video Ctrl
RIVA128?

Sound

Local Mem.
(SRAM?)

Memory
Controller
(FPGA)

SCSI-I/F

Cache
Memory

PCI-PCI
Bridge
(FPGA?)

Sub MPU
MCF5307
(ColdFire)

Bus Sync.
(FPGA)

Q-Bus
Arbitor
(FPGA)

USB Ctrl'er

Local Mem.
(DRAM?)

Interrupt
Controller
(FPGA)

PCI Slot
(2 Slots)

FIFO Ctrl
(FPGA)

Q-Bus Slot
(2 Slots)

Q<->SSBus
Bridge
(FPGA)

Universal I/O
(Joy Stick)

FIFO Mem.
8K×64Bit

DMA
Controller

SSBus
(X/4 MHz)

Super-I/O
Keyboard
Mouse, etc.

SDRAM
Controller
(FPGA)

IOCS-ROM
(2MB)

SS-Bus Slot
(2 Slots)

SDRAM
(X MHz)

Universal
Timer(4ch)

System Ctrl
LED/Sw..etc)

ませんから，種類の違うMPUを動作させる場合には，オブジェクト(実行ファイル)も動作させるべきMPUに合わせてリコンパイルしたものが必要になります。

OSはマイクロカーネル指向ですから，カーネルの機種依存部分さえなんとかすれば，ほかのMPU用のOSとすることは，そんなに難しいことではなくなります。先にも書きましたとおり，各MPU用のコンパイラを用意すれば，即マルチMPU環境になるように設計します。ソフトウェア関連については，別項で触れることにします。

今後も，零式の開発状況に関しては，ホームページなどでもご紹介していきます。

・SDRAMハイメモリ

零式では，販売価格を低減させるため，その開発にかかる初期投資費用(イニシャルコスト)を削減しようとしています。

その意味では，九九式試製一型の原価販売も，イニシャルコスト分散の一環としたいのですが，それだけではまかないきれません。

そこで，零式に搭載される予定のチップを，X680x0関連製品として活かしてしまおうという計画を立てました。そのひとつが，SDRAMハイメモリです。文字どおり，X68030とXellent30用にもSDRAMで構成されるハイメモリを製作しようという計画です。いわゆる派生商品です。

ボードに載るメモリは128Mバイトですが，もしかすると64M版も用意することになるかもしれません。ボードの外形は，以前ツクモさんが出しておられた，いわゆる元祖ハイメモリを想像してください。価格は45,000円前後になると思います。

SDRAMにより高速なアクセスが可能になるはずです。理論的には2-1-1-1アクセスが可能なはずなのですが，X68030/Xellent30側が遅いため，これは実現しそうにありません。もっとウェイトが入ることになるでしょう。

この原稿を書いている時点では，すでにアートワーク作業に入っています。

・PCカードインタフェイス

X680x0用に，個人ベースでPCカードインタフェイスを作っている方がおられます。これを製品化しようということで，企画がスタートしました。最初はISA-PCMCIAのブリッジチップを導入しようとしたのですが，ものすごい条件が出てきたため(最低発注数が単価2,500円で480個！まったく人をナメた話です)，まさちく工房さんに，ブリッジチップの内製を依頼することになりました。製品化までは少し時間がかかるかもしれませんので，気長にお待ちください。

なお，拡張スロットに差すタイプの製品になりますが，カードスロットは，フラットケーブルで外部に引き出す仕様にします。

・Mach2p

SCSI-2ボード(Mach2相当)と10MB拡張メモリボード，PSX16550相当の高速RS-232Cボードの複合ボードです。PROにも「一応」対応しますが，残念ながらPROではバスマスタ転送ができませんので，ソフト転送のみとなります。

当初はEthernetも載せる計画でしたが，実装に時間がかかりすぎるという理由から削除になる見通しです。遅れに遅れています。面目ないです。SDRAMハイメモリとどっちが早いか……といったところですか。

・Jupiter-X

ACE/EXPERT/SUPER/XVIへ68060を載せてしまおうという豪快な製品です。メモリスロットを3つ持っていますが，うち1スロットは16Mバイト専用で，ほかの2スロットは128Mバイト用です。

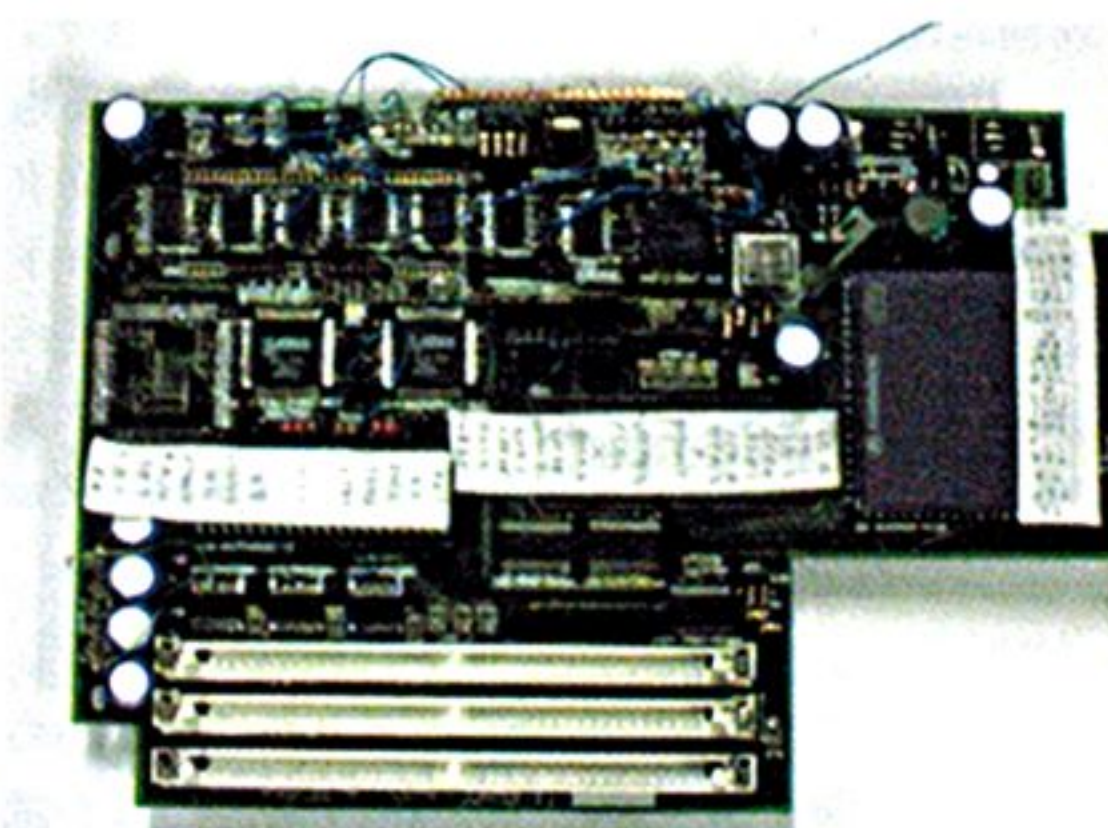



図3 Jupiter-X

なお，これらの製品では珍しくDMAがハイメモリ領域に届く設計なのですが，対策部品の入手が思わしくなく，これも遅れています。夏頃の発売を目標にしています。

・4倍速スキャンコンバータ

この製品の概要をご存じでない方のために解説しておきますと，この製品は，15kHzで入ってきた画像(RGB/NTSC)を，31kHzないし63kHzにアップスキャンするコンバータです(ですから，正確には「4倍速スキャンコンバータ」となります)。

やっていること自体は，基本的には走査線のコピーですが，動き適応ができるようになっています。これができるとなにが嬉しいかというと，静止画で解像度を落とすことなく鮮明に表示できるほか，通常のインタレース画像では，動画にギザギザ感が出ることがありますが，これを極力抑えることができます。

また，フィールドスキャンも行えます。ラインスキャンでは前後の画像は参照しませんが，ひとつ前の画像と現在の画像を，ひとつの画像(ノンインタレース)に合成するのがフィールドスキャンです。

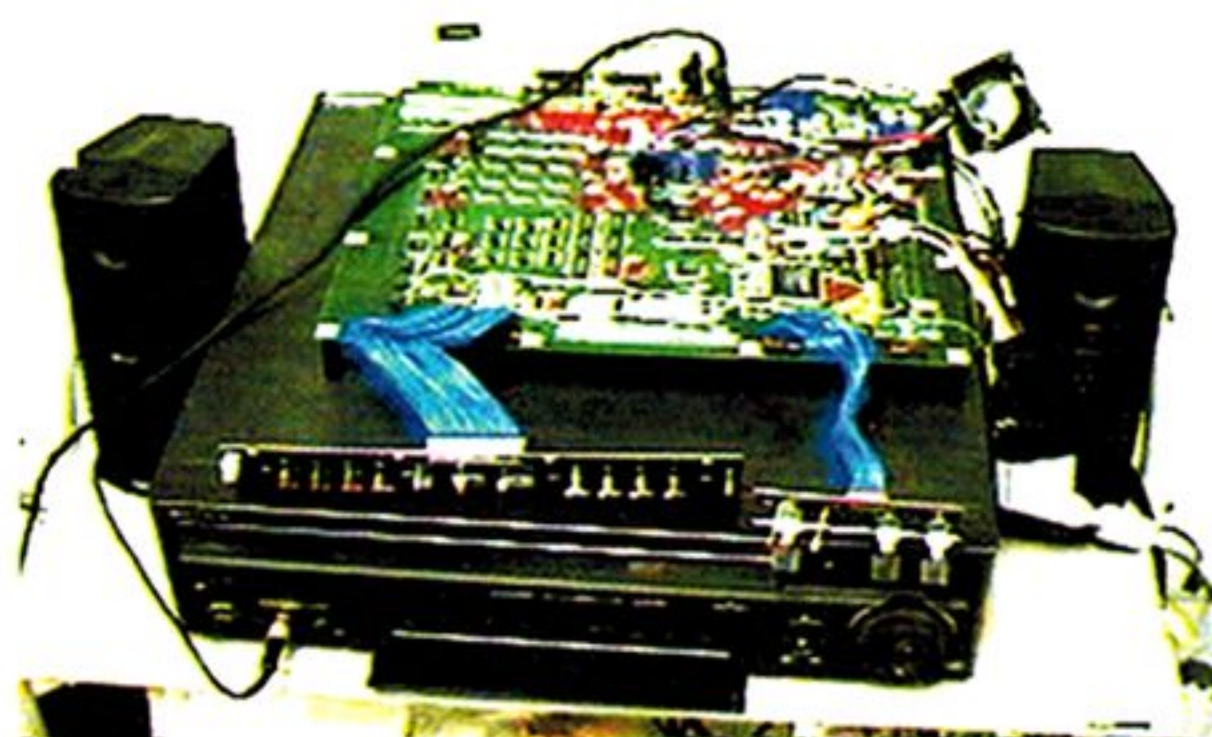
X680x0ユーザーにはさらに特典があります。キーボードコントロールができるので，マルチスキャンモニタを使用している方には福音となるでしょう。さらに，IMAGE端子による取り込みにも対応できる見込みで，30ピンハーフピッチアンフェノール端子が標準装備されます。

バーチャル15モードも装備しています。これは，アップスキャンしているときにも，15kHzのような溝の埋まっていない画像にできるというものです。そちらの画面モードが好きな人にはたまらない機能でしょう。個人的には，4倍速の，ラインがみっちり埋まった画像が好きなのですが。

入力は15ピンRGBと21ピンEIAJRGB端子，S端子，ビデオ端子の4種類です。最後に入力された信号に，自動的に追従して切り替わるモードもついています。また，オートパワーオフも備わっています。

図4 4倍速スキャンコンバータ

零式は趣味のマシンだ

column

自動車を例にとってみましょう。古すぎてメーカーにさえ補修用部品がないような車を，わざわざ部品取り用に2台買ったりして，修理しながら乗っている人がいます。さらに突き詰めていくと，自分で自動車会社を作ってしまった人もいます。

パソコン界もそれに似た状況があってよいのではないでしょうか。車にたとえると，現状では貧弱なシャーシに巨大なエンジンを載せているようなもので，私にはちょっと偏っているように見えます。それでも楽しいのであれば，私ごときが口を挟む領分ではないでしょう。ですが私は面白くありません。

私(私たち)も，「自分がほしいパソコン」を作りたいがために，世界的な水準からいっても無茶な計画を立ち上げました。それが零式計画です。

これを「究極の自己満足だ」「でっかいおもちゃだ」「技術屋のオナニーだ」という人がいるかもしれません。でも，我々は命をかけて遊ぼうとしているのです。それでいいのだと考えています。

出力は，15ピンRGB端子と，BNC端子の2つが使えますが，両方とも同じ信号が出力されます。

現在，1号機が調整の最終段階に入っており，2号機・3号機の製作と筐体の設計も始まっています。もうしばらくお待ちください。

・X68030用拡張スロット

前号でもちらりと書きましたし，コラムでも触れていることですが，TSRさんから出ると期待されていながらも，とうとう出なかった，いわば「X68030用XpanderIV」を作る計画です。

これ自体は，コネクタの整合さえ取れればすぐにでも着手できる内容なのですが，やはりものには順番というものがあり，ほかの計画を終わらせてからでないと手をつけられません。

現状では，側板の型が残っているかが不明です。もし残っていない場合，コストの点から型を起こすことは不可能ですから，側板なしで販売することになるでしょう。Xpander IVでは強力な電源ユニットが搭載されていましたが，側板なしでは危なくてそのようなことができませんので，ACアダプタ給電式に改修するなど，独自の改造が必要になるものと思われます。このあたりに目鼻がつくまで，もう少し時間がかかるでしょう。

各種キット

こちらもADCキット，同軸・光端子変換キットなど，出さなければならないキットがたくさんあるのですが，まだ手がつけられないでいます。デジタルボリュームキットなどの要望もあり，こういった先鋭的な，というか，よそがやらないようなキットをどんどん出していきたいと思っています。

・光DAI出力化キット MK-CC003

MK-CC003「光DAI化キット」は，その名のとおり，デジタルオーディオを内蔵している機器から光デジタルオーディオインタフェイス(DAI)を無理やり出してしまおうというキットです。これを装着すれば，PlayStationやDreamcast，SEGA SATURNなどの音声を劣化させることなくデジタルのままDATやMDに採れてしまいます。おかげさまで大受けで，一時は生産が間にあわなかったほどです。生産といっても，基板以外は家内制手工業に近い手作業なんですが。

なお，一度に3台や4台をお買い求めになるお客様も多く，「やっぱりスキモノな人はいるんだなあ」と一同ほっとしているところです。

ところで，デジタルオーディオならなんでもいいかというと，そんなことはなくて，たとえばサンプリング周波数は32/44.1/48kHzに限られますし，受け手の問題として16ビットでなければ再生ができないという問題があります。ですから，有線放送機器やMIDI楽器は，改造してもうまく録音できないケースもありえます。MDによっては，サンプリングレートコンバータを搭載していない場合，32/48kHzの録音ができないこともあります。

なお，いうまでもないことですが，著作権法の規定により，録音は個人的使用に限られますので，ご注意ください。

・HC000変換基板 MK-CC004

モトローラから20MHz版のMC68HC000が発売されているのですが，残念なことにPLCC版しかありません。PLCCというのは，足がゆでだこのように内側に回っている小さなパッケージのことです。MC68030RCなら，同じパッケージで高クロック版が存在するのでよいのですが，この場合はそういうわけにはいきません。

そこで，クロックアップをしている人が安心して暮らせるように，PLCCパッケージをシュリンクDIP(元からの形状)に変換する基板を作りました。最初から20MHz版を搭載したものも用意しています。数に限りがありますので，お求めはお早めに。

・倍クロック生成基板 MK-CC006

これは040turboのユーザーさんを意識して作った，完全に趣味のキットです。が，使っている石がサイプレス社のCy7B991なので，ついているデータシート(英文)を読みこなせば，スキューの調整とか結構マニアックなこともできます。ちょっと抵抗の飛ばし方がアレで恐縮なのですが。

・ATX電源接続キット MK-CC011

X680x0が故障して動作しなくなるときは，たいてい電源ユニットが故障しているケースがほとんどです。

さて，最近のAT互換機はATX規格がほとんどですが，電源ユニットがやっとインテリジェント化(といっていいのか？)されました。つまり，常に生きているサービス用の5V系(+5Vsb)があり，電源ON信号があるタイプの電源ユニットが，安価で市販されるようになったわけです。これを見逃す手はありません。

ATX電源接続キットでは，X680x0のPC信号をATX電源のPS-ON信号に変換(といっても，論理を反転させているだけですが)して，X680x0の前面電源スイッチに連動させるようにしています。

なお，ATX電源用のコネクタは備えているものの，コスト削減のため，X680x0へのコネクタはキットに入っていません。どうせ電源ユニットが死んでいるのですから，そこから切断して使うという仕様になっています。これなら，コネクタが合わないということはありえません(ケーブルの長さが足りないことがありますが)。

ですので，もしお手元に電源が壊れたとおぼしきX680x0があれば，このキットとちょっと弱めのATX電源を買ってきて，復活するかどうか試してみられてはいかがでしょうか。ただでさえ貴重になりつつあるX680x0が蘇るかもしれません。

ところで，ちょっと弱めのATX電源でも，いきなり250Wクラスになるようです。もとからのX680x0電源の3～5倍のパワーがあり，誤接続は即命取りですので，くれぐれも注意して組み立ててください。

・連動電源キット MK-CK002

これは，X680x0だけでなくほかのマシンにも使えます。X680x0の場合は，ジョイスティック端子の+5Vを見張っていて，SSRで電源を連動させるキットです。電源タップ付き。

締めにあたって

夏頃にかけて新製品ラッシュ状態で，体がいくつあっても足りない状態が続きますが，X680x0ユーザーさんと零式に期待してくださっている方々のためなら，キーボードを枕に討ち死にする覚悟でやっていきますので，どうかよろしくお願い申し上げます。

末筆になりますが，まーきゅりーゆにっとと，X68030用内蔵メモリMK-5BE8の在庫がだぶついています。まだお買い上げでない方は，この機会にどうかお買い求めください。以上宣伝でした。

図5 ATX連動電源キット

TSRさんとの業務提携について

column

満開製作所では，このたび，東京システムリサーチさん(以下TSRとします)と次のような合意に達しました。

・TSRさんのX680x0関連製品に関する事業を，満開製作所に譲渡する。
・その代わり，満開製作所は，今後TSRさんのX680x0関連製品のサポートを引き継ぐ。

この件ではすでに資料を引き継いでいます。ですから，今後TSR製品をお使いの方で，ご相談のある方は，満開製作所までお願いします。とはいえ，TSR製品は優秀なものばかりですから，基本的にないとは思いますが。

また，事業移管により提供される資料を活用して，TSRさんのノウハウを，満開製品に生かしていくつもりです。とりあえず，別記したとおり，拡張スロットを4スロットに増やすXpanderIVのX68030版を製作しようと考えています。

さらに，XSIMMシリーズの回路を満開製作所の製品で活かすことや，Xellent30シリーズを上位MPUで展開することも考えています。

零式ＯＳ構想

満開製作所ソフトウェア開発部
鎌田誠　Kamada Makoto

零式のソフトウェア環境が目指すところをひと言でいうと，「プログラミングを楽しめる環境」ということになります。プログラミングの勉強ができて，しかもユーザーが十分楽しめる程度のゲームなどをユーザー自身が作れるような環境です(そのために必要な知識と道具と材料になりうるものはできる限り標準で添付します)。

このようなソフトウェア環境を支える零式標準OS(オペレーティングシステム)を，満開製作所は独自開発で提供することにしました。既存のOSを採用してもよいのですが，前述のようなソフトウェア環境を構築するのに適しているOSが見当たらなかったのです。もちろん，一般に広まっているさまざまなOSのインストールを拒むわけではないので，パワーユーザーであれば「OSの移植」というプログラミングの楽しみ(?)もあります。

零式のOSは現在構想段階で「やっとまとまりつつある」といったところです。今回は零式のOSに欠かせないと思われるいくつかのキーワードを挙げておきます。

マルチタスク/マルチスレッド方式

零式では開発環境も快適にしたいので，最低限，エディタとコンパイラは同時に動いてくれないと困ります。当然，Human68kのバックグラウンド処理機能のような中途半端なものでも困ります。

そこで，零式のOSはマルチタスクに対応し，タスクごとに論理アドレス空間が完全に分離されるものとします。また，ひとつのタスクに複数の実行権を与えることができますが，同一タスク内のスレッドは同じ論理アドレス空間を共有するものとします。

タスク同士の情報のやり取りには，メッセージパッシングを主とするタスク間通信を使います。大きなデータのやり取りはページングを使って高速化します。

マルチタスク動作の安定性を考えると，必然的に，「すべてのユーザータスクをユーザーモードで動作させる」ということになります。プロセッサの保護レベルが多ければこの限りではないのですが，とりあえず最初に載せるMC68060には保護レベルがユーザーモードとスーパーバイザモードの2種類しかないので，いたしかたありません。

それではユーザータスクがI/Oを直接叩くことができないのかというと，そうでもありません。I/Oの叩き方については，おいおい説明したいと思います。

リアルタイム指向

開発環境だけでなくゲームなどの実行環境もあわせ持つには，マルチタスク/マルチスレッドに加えて「リアルタイム性」という要素も外しがたいものです。

X680x0では，プロセッサ(MC68000など)の動作速度やOS(Human68k)の機能がとても低いにも関わらず，かなり手の込んだシューティングゲームを作って動かすことができます。これにはハードウェアスプライトの効果もありますが，ソフトウェアの面では，水平帰線，垂直帰線，あるいはキー入力といった割り込み処理をユーザーが自由に記述できるという点が重要です。これらの割り込み処理を「プリエンプティブなコンテキストスイッチによって実行されるスレッド」と解釈すると，それはリアルタイムOSのスレッドによく似ています。それをどれだけ簡単に記述できて，どれだけ高速に(この場合は平均速度ではなくて応答速度)処理されるかが問題なのです。

零式のOSでも，割り込みをトリガとする優先順位付きのコンテキストスイッチを採用します。ただし，ユーザータスクはすべてユーザーモードでなければならないので，ユーザーが記述できるのは割り込みルーチンそのものではありません。

マイクロカーネル指向

普通のUNIXのようなモノリシックカーネルにしてしまうと，OSの拡張性に乏しくなり，また，デバッグも大変です。そこで，カーネルをマイクロカーネルにします。つまり，「必ずしもカーネルの内部になくてもよい機能」をカーネルの外に押し出して，カーネルのサイズを小さくするのです。

マイクロカーネルにすると，モノリシックカーネルではカーネルの中にあったものをユーザータスクで提供することになります。そのためコンテキストスイッチやタスク間通信が増え，モノリシックカーネルと比較して動作速度が低下する心配があります。しかし，ここはOSの保守性を優先したいと思います。

以上の説明だけでも，零式OSがHuman68kとはかなり異なるものであることがわかると思います。「Human68kのような使い勝手でないと嫌だ」という人もいると思いますが，マイクロカーネルの上に載せるサーバやライブラリによってHuman68kに近い使い勝手を生み出すことは可能だと考えていますので，ご心配なく。

今回書いたことは，零式のOSの構想のもっとも基本的な概念の，しかも抽象的な部分だけです。より具体的な構想については，次回以降に書きたいと思います。

日本語入力関係は……

零式を巡るソフトウェアのなかでも，かなりの人が心配しているのではないかと思われるものに日本語入力関係がある。互換機なのだからとASK68Kをそのまま動作させるだけだと，たぶん互換モード(?)での動作にしかならない，ということになるのではないかと思われる。

なんとなればUNIX系のものを移植してくるという手もあるのだが，それらのシステムのものがさして賢かったり，使いやすいわけではないというのも問題だ。ASK68Kがさほど高性能ではなかったというのはユーザーなら誰しも知っているだろうが，恐ろしいことに世の中にはそれよりひどい日本語システムのほうが多いという事実があるのだ。おおくのプログラムでは非常に高度な変換プログラムを搭載して，非常に高度な使いにくさを実現している。

日本語変換関係だとフリーウェアが少ない，という点と，さらに問題なのは軽くて使いものになるプログラムが少ないという点だ(フリーであるなしに関わらず)。かといって作るのは楽なことではない。

ASK68Kはアクセスとシャープが制作した日本語変換システムだ。辞書とかは勝手にいじって，すでに原型を留めていないくらいにまで改変したのだが，変換エンジンはどうしようもない。辞書をいじることでASK68Kは相当に使いものになるFEPとなる。しかし，本体もいじればもっともっとよくなるはずなのだ。

ASKには問題点もある。ひとつが辞書容量の制限だ。辞書容量はすでに限界(1MB)に達している。個人的にさらにもうひとつ挙げるとキーバインドの制限である。あと，ついでに何点か変換部分がいじれれば，別に高度なものなんかいらないと思わせるくらいのものにはなるだろう。そんなWindows版があったらすぐに買うのだが。

というわけではないが，現在ASKの改良版を作っている人もいる。辞書の容量制限を4MBにまで拡大したテスト版が作られている。1MBから4MBということで，必要十分な容量だ(やってみれば100KB足すのも結構大変だということがわかるだろう)。

辞書をさらに完全オリジナルベースで作り(すでに辞書構造はASKとは異なる)，変換部分もオリジナル色を強くしていけばかなり見通しは明るいのではないかとも思えるのだ。それとも楽天的すぎるだろうか。　(U)

X68000だけの催し物をやりましょう！ フェスタ・68へ行こう！

column

1998年は，ここ最近でのX68000最良の年でした。満開製作所はNew-X「零式」の開発を発表し，この「Oh!X」はまさかの復刊を果たしました。現時点で，あとX680x0に足りないものはなんでしょうか？　まだまだたくさん足りないものはあると思いますが，これからさらにX68000を盛りたてるのにいちばん必要なもの，それはX68000だけの催し物です！

ということで，インターネットをはじめとする通信環境を活用していないユーザーのために，X68000がメインのイベント，「フェスタ・68」を企画しています！

開催日は5月4日。場所は東京千代田区外神田にある「マーク・インタースペース」というビルの1フロアを借ります（東京都千代田区外神田6-14-8。銀座線末広町駅より0分，千代田線湯島より5分，JR・地下鉄秋葉原駅から徒歩7分）。開催時間は午前10：00～午後3：30という日程になります。

フェスタ・68で提供するのは展示即売スペースとゲーム大会などです。誰もが楽しくX68000と戯れられる場を設けることを目標としています。この会場は電源も使えますので，各サークルの方のデモや怪しいものなどもいろいろ見られるのではないかと思います。

現在の出展予定はハード，ゲームソフト，CDマガジン，グッズ関連が多いのですがX68000のソフトに飢えてる方や，これからX68000を使われる方にはきっと満足のいく催し物になることと思います（現時点での参加予定サークル一覧を参照してください）。

ここで具体的な展示内容の例を挙げますと，個人レベルで開発されてるハード（040Excel，PCカードスロットなど）のデモンストレーションや，ひょっとしたら満開製作所が製作中の九九式が展示されるかもしれません。現在もサークル参加を募集していますので，希望される方はこちらまで！

http://www.pipi.net/X680x0/

また，一般参加者の方への情報です。現在，「チケットぴあ」より入場券（300円）が絶賛販売中となっています。当日の会場は混雑が予想されますので，できるだけこちらをご利用ください。

出展予定者一覧

企業参加 ：満開製作所
出展内容 ：同人誌・ハード・ソフト
　九九式，4倍速SCANコンバータ，Jupiter，CD-ROMなど

サークル参加 ：GAMEAGES
出展内容 ：ゲームソフト

サークル参加 ：TAIAN
出展内容 ：アダルトゲーム

サークル参加 ：MAT研究所
出展内容 ：X68ハード＆ソフト＆データ
　ispr動画RealPCM対応バージョン
　TS-6BGA　PCMオンリーボード

サークル参加 ：D.O.J.
出展内容 ：SFXVI

サークル参加 ：D.C.S.
出展内容 ：SFXVIデータ他

サークル参加 ：XPS
出展内容 ：データー類

サークル参加 ：Colorful Tips
出展内容 ：ソフト類

サークル参加 ：Team Dangeroude
出展内容 ：ソフト類

サークル参加 ：妙ゲーム
出展内容 ：ソフト類

サークル参加 ：すたじおRE･O･NA
出展内容 ：040Exee Iのデモ

サークル参加 ：ION PROJECT
出展内容 ：ソフト類

サークル参加 ：TEAM CATHEDRA Project
出展内容 ：ゲームソフト

サークル参加 ：TSR SOFTWARE
出展内容 ：同人誌＆ハード類

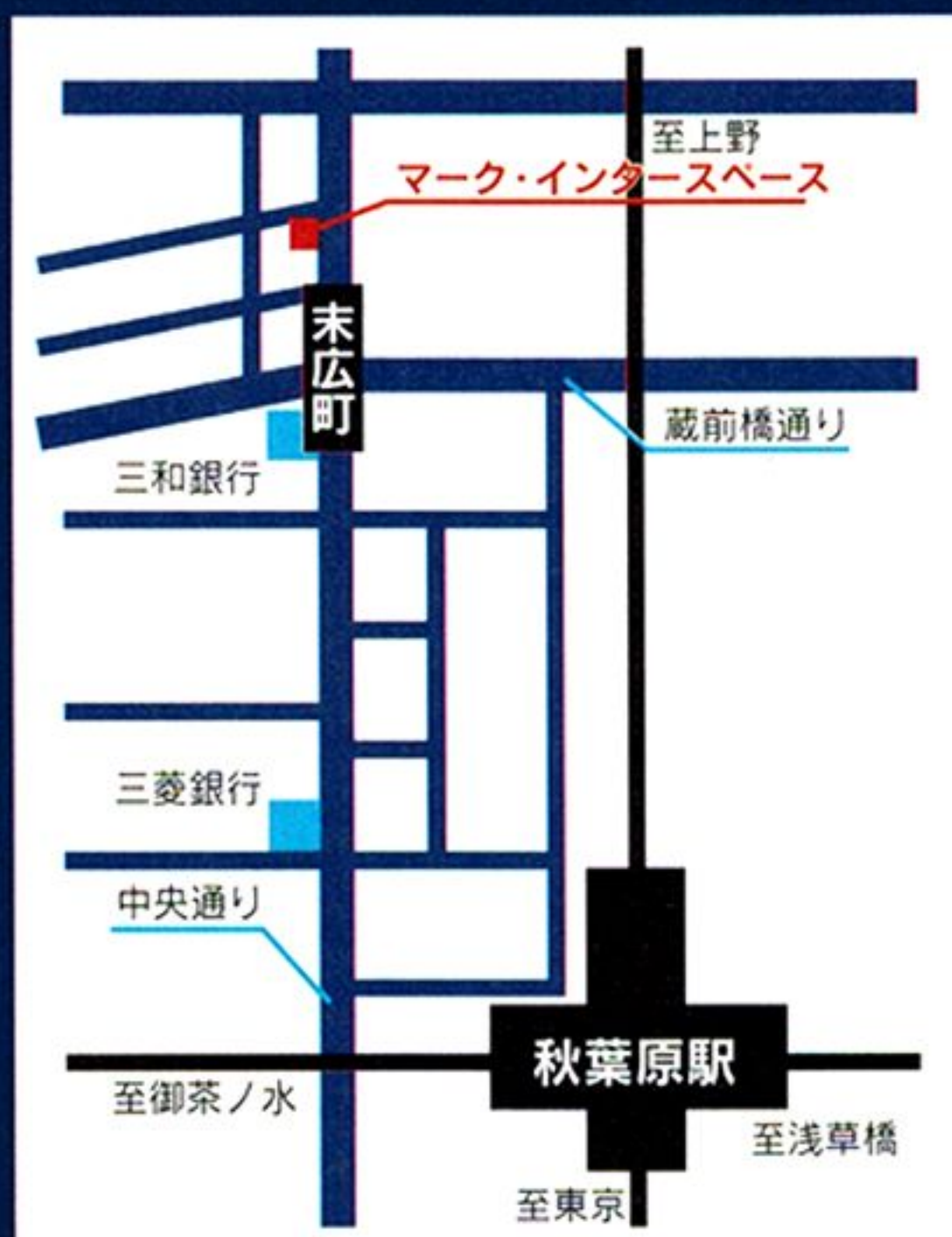
サークル参加 ：「めるほぬ研究所」
出展内容 ：PCカードスロットの展示

サークル参加 ：「OffsideX680x0」
出展内容 ：「OffsideX680x0」68用ディスクマガジン

サークル参加 ：Ran2
出展内容 ：SFXVI関連のデータ集

サークル参加 ：超連射68K
出展内容 ：シューティングゲーム

サークル参加 ：東京電脳遊園地
出展内容 ：CD-ROMマガジン、PS2マウスコンバータ

特集 ネットワークゲームの地平へ

「ネットワークゲームの時代がくる」
というのはひとつの神話ののようなものだった。日々の電話代などのことは忘れ，いつかはワイヤーの向こうに広がる広大な遊園地で思う存分遊べるようになる，という淡い期待をいだいてる人は少なくないだろう。さほど荒唐無稽な話ではない。
ネットワークゲームに託されていた夢のひとつは，UltimaOnlineがあっさり達成した。そして，DirectPlayなどで比較的簡単にネットワーク対応にできるようになってきた。海外産のゲームを見れば「とりあえずネットワーク対応しました」という感じのものがどんどん増えてきた。一時のPlayStationソフトの「とりあえず3次元にしてみました」的なものと同じような感じであらゆるソフトがネットワーク対応になってきている。
海外のこのような積極的な動きに比べて，国内でのネットワークゲームの展開は遅々として進んでいない。通信環境などの問題があるにしても現状ではさまざまな問題は一応クリアされてきていると見ていいのではないだろうか。その気になれば，OCNなどの個人で導入可能な低価格専用線環境，低価格サーバマシン，DirectPlayなどの手軽な開発環境などが一応整ってきているのだ。「小規模であれば」という条件はつくかもしれないが，一時はいくら望んでも得られなかったものに手が届くようにもなってきているのだ。その先に広がるのは未踏の領域である。
やがてくるであろうネットワークゲームの大波に備えて，いまのうちに予備研究をしておこう。

CONTENTS

176	：認識の境界を超えて	中野修一
180	：多体問題と相互作用を考える	三沢和彦
186	：Diabloに見るネットワークゲーム構成法	瀧康 史
192	：インターネットセキュリティ事件簿	三沢和彦
194	：インターネットゲーム時代のコミュニケーション論	古村 聡
198	：海外コインオペゲームにおける通信	市川幹人
200	：アーケードゲームに見る通信対戦の未来	如月 緑
205	：DirectPlayを使った対戦ゲーム	菊地 功
210	：Javaネットゲー考察 ExpertMission制作編	霧雨
216	：ExpertMissionユーザーズマニュアル	野沢絵美
220	：CGIによる継続したユーザー管理	大和 哲
224	：Macintoshをサーバにしてゲームを作ろう	古籏一浩
228	：ネットワーク対戦エミュレータ“Bot”	須藤芳政
232	：第2の人生，Ultima Online	kazuhisa@Pacific

SPECIAL

特集 ネットワークゲームの地平へ

認識の境界を超えて

中野修一 *Nakano Shuichi*

ネットワークだとなにが違うのか，そもそもクリアしなければならない問題はなんなのか，そしてどのような対策が取られるのか……。やがてくるであろう「ネットワークゲーム当たり前の世界」の姿を垣間見てみよう。

世の中はネットワーク真っ盛りだ。パソコンだと，いったいみんななにやってるんだろうと思うくらいインターネットが普及しているし，Oh!Xのハガキとか見ているとDreamcastでインターネット始めましたという人も予想以上に多いようだ。なんやかんやで，

「次はネットワークゲームだよね」

ということになっているらしい。

そろそろ，

「ネットワークでなにができるか，ちょっと考えておいてくれないか」

とかいわれて頭を抱えているゲーム屋さんもいるかもしれない。

「インフラ揃ってねーよ」

といっても，そのうちやらなきゃいけないテーマであるのはたぶん間違いないので，逃げて回っているわけにもいかない。たぶん，先に手を出したほうが勝ち，とまではいわないが，十分に有利なのは疑いようがない。この業界，そろそろ正念場……という感じの方もいるのではないだろうか。

では，ネットワークゲームとはどのようなものか？　成功例としてUltimaOnlineなどを見てみる。確かに凄い。大きなサーバと潤沢な回線が必要だと結論する。しかし，

「予算はない」

ということもありがちかもしれない。

回線は遅い，速度は安定しない，通信コストは高い，サーバ設置などの予算はない……。などという状況が目に浮かぶようだ。

■ゲーム以前の問題点

日本でネットワークゲームを立ち上げるには，通信インフラの問題がもっとも大きい。その点では次世代ISDNが本命となるだろう。いまのところ，月額3500円で128kbpsでの常時接続が可能ということになっている。現状のモデム接続の3倍程度の速度になる。将来的にレスポンスが3倍よくなることを仮定したとして，現状でできる範囲のことでノウハウを蓄積しておくというのは悪いことではない。おそらく接続環境としてはこれ以上のものを期待するのは当分難しく，また，この接続環境でちゃんと動かないようなものを作ってもしかたがないと思われる。

これが具体的にいつから稼働するのかというのは不明だが，いまのISDN回線の設備のまま常時接続環境に移行できるということで，局側の対応だけが問題なのかもしれない。その準備ができれば比較的早期に実現される可能性もあるのだ。インフラについては，一応見通しは明るいと思っている。これがポシャっても代わりのものは出てくるだろう。固定料金の常時接続環境は時代が要求しているものなのだから。とにかく，そろそろGOサインを出してもいい頃だ。

次に問題になるのが「どうやって金を取るか」だ。ビジネスとして成立させるにはお金を取らないといけない。個人が趣味でやるなら関係ないんだが。

日本人のメンタリティではサービスに対して金を払うということを期待するのは難しいとよくいわれる。確かにそうなのかもしれない。ゲームプレイヤーの多くが未成年と推定されるようなコンシューマ関係だと，クレジット決済，特に従量式の課金は難しい。

また，サポートの苦労を理解している人なら，オンライン化されることによってそれが何倍にも膨れ上がると思っていいだろう。おそらくトラブルの発生率は上がり，メーカーへの連絡はつけやすい。ゲームの維持管理だけでもかなりのコストが発生するのに対し，コストの回収は難しい。

パッケージ価格に含める
広告を入れる
会費制とする

電子マネーなども時代の要請ではあるのだろうが，NTTが頑張れば大半は解決する回線の問題よりもおそらくずっと難しい問題だろう。

当面のところ，費用の回収を考えるのは無謀だろう。実験用ということで進めて，十分な体制が整ったら課金制以降ということだろうか。もちろん課金した途端にサーバから人が消えるという心配もあるのだが……。ま，内容と課金のしかた次第だ。

タダであっても「タダだからサポートしなくていいと思ってるのか！」とか「まあ，タダなんだから我慢しなくちゃいけないんですよね」とかイヤミっぽいことをいわれなきゃいけないご時世だ（ジオシティーズを参考にしました）。これで金を取っておいてサーバが落ちたとか，キャラクターデータが消えたとかになると，どんな騒ぎになるかわからない。以前は，インターネット接続プロバイダのディスクが飛んで，騒ぎになったこともあったが，そういう時期にインターネットなんてものに手を出していたような人だったら，ある程度内情も察してもらえて理解も得られただろうが，最近の，箱開けて5分でインターネットにつなぐような人たちに同じことを期待するのは難しいかもしれない。ページは開いて当たり前，メールは瞬時に届かないと異常だと思っている人たちも多いのだ。

まあ，このあたりは考え出すと別の時限の話になるので今回は扱わないことにする。

■お茶の濁し方

とりあえず，本格的なものの前に軽くできそうなものから検討していってみよう。

回線の問題などを考えると，まともにやるのは費用がかかりすぎそうだ。個人レベルでできる話でないと面白くない。別にOh!Xはその手の業界誌というわけでもないんだし。

もっとも，どんなにいい回線を使っていてもやらなければならないこと自体は，ほとんど変わらない。たとえ10秒に1回が1000秒に1回になったとしても，エラー時の処理は実装せねばならないわけだし，クリティカルなタイミングで回線切断された場合の対処もどうやったとしても省略することはできない。送られてくるデータの時系列の保証すら疑わなければならないかもしれないのだ。というか，万一のためのフェイルセーフ機構はほぼあらゆる部分に必要になってくる。

小規模なりとはいえ，まっとうなシステムが立ち上がれば規模を大きくすること自体は単にお金だけの問題に集約できるだろう。

回線切断，サーバダウンなどは常に考えられる事態である。そういったときに落ちるのはしかたない。大事なのはそこで不都合を起こさないことだ。ほら，ちょうどそこ（DOS/V magazine編集部）に，丸1日キャラクターを育てて，変なタイミングでサーバが落ちちゃったので真っ白になってるのが約1名。キャラクターさえ残ってれば許してもらえるんだけど，間違って消えてたりしたら，大変だろうなあ……。

ということで，比較的簡単そうな実装方法を順に見ていこう。

●遠隔対戦

いちばん簡単な方法は，マルチプレイヤーのゲ

ーム（対戦ゲームなど）を既存のゲーム内容のまま，遠隔対戦というフィーチャをつけて「ネットワーク対応」としておけばいい。基本的に1対1の接続で，アクセスサーバはロビー処理とマッチメーキングだけえお担当する。ゲームプログラム側は多少変更が必要なので，これだけでも十分大変なんだけど，比較的労力は少ない。

マッチメークできなかったときのために，コンピュータプレイヤーが動くか，センターの人員が相手をするかとかで多少アレンジは可能だろう。また，ランキングや掲示板で一定のコミュニティを形成していくことはできるだろう。常連客を作ることは重要だ。たいていの場合はメンテナンスまでやってくれる。大々的な盛り上がりを期待するのは無理だろうが，比較的小規模でネットワークゲームの実験などを行っていくことはできる。

●非リアルタイムゲーム

比較的小規模な回線とサーバでできる範囲のことを考えると，通信遅延が気にならないゲーム構成を最初に作ってしまえばいい。60fpsのゲームが1fpsになると目も当てられないが，1ターンが5秒程度のゲームなら，それが6秒になってもあまり気にならない。

●準リアルタイムゲーム

だいたい人間の知覚からして，少々遅くてもある程度は補間して感知できるので，フレームレートが十分にない場合でもリアルタイム的な展開は可能になる。

「1/60でないとゲームじゃない」

という人もいると思うが（そういう人が結構平気でプレステのゲームとかやってたりするのだが），現実問題として，

「ネットワークゲームだから」

ということで許される部分は多いと思われる。

甘えられるところは甘え，妥協できるところは妥協して，それでもまだ速度が足りないかもしれないというのが現状だろうから，仕様は割り切って考えたほうがいいだろう。

通信速度だけがボトルネックになるなら，それ以外の部分では思いっ切り重い処理をしてもボトルネックになることはない。たとえば，データ圧縮をして転送するとして，ハフマン符号化などというケチなことはいわず，算術圧縮符号化を行ったり，地形などはフラクタルモデルから自動生成したりという感じだ。

どうあがいても1秒間に5回しか画面更新できないなら，1画面の描画に0.1秒かかるような重いシーンを連続してもまったく問題がないということになる。画面の情報量を極端に上げることもできなくはないだろう（もっとも，それだけオブジェクト数が増えて通信の負荷になる可能性もあるのだが）。PCだと性能が一定しないので突き詰めることは難しいが，ドリキャスやPS2とかN2Kとかになれば，リアルタイムゲームとは別次元の高密度な映像を展開できる可能性がある。

以上のような展望から可能なゲームデザインを考えてみよう。非リアルタイムゲームはある意味で「なんでもあり」なので割愛したい。対人対戦の代わりも面白くないな……。結局残るはひとつか。無論，リアルタイムでいければそれが理想だが，接続環境設定メニューのいちばん下に「T3」とかがあるのではユーザーが限られすぎる（でもそういうのもちょっと，いいかも）。

ただ，完全非リアルタイムの場合でもターン制といっても思考型のゲームをネットワークに持ち込むのは危険だ。ひとり当たりの思考時間が長いものを多人数でやるというのは不可能とはいわないが難しい。時間にクリティカルではないけどサクサク操作できるものが無難だ。将棋並みに長考の可能性があるものだと受け入れられる人がかなり少なくなってしまうだろう。

さて，準リアルタイムゲームの場合，時間管理をどのように行うかがまず問題になってくる。ターン制のように同期するか，非同期でいくか。ある程度以上に参加人数が多くなるゲームだと同期式はつらい。いちばん遅い回線がボトルネックになってくるからだ。全体の速度はそれに支配される。かといって，非同期だと「回線が速くてマシンも高速でないと生き残れない」とか「いつのまにかアイテムがない」とか，いったことも起こる。リアルタイム性優先のゲームの場合は，これもある意味公平な処理の結果なので認めざるをえないだろうが，モデムだとゲームにならないというのではちょっと困る。

「準」リアルタイムでいいと割り切ることで，リアルタイムのように時間進行優先というわけでもなく，非リアルタイムのようなユーザーの処理優先というわけでもない，独特なゲーム時間の管理が必要になってくる。

ネットワークゲームに限ったことではないが，ゲームの内部の時間は連続的には流れない。量子的な時間をうまく管理してやらないといけない。ここで別に時間の離散性が強くなったとしても基本処理はあまり変わらない。もともとゲーム内の時間は離散的なものだから。1/60秒単位で動いていたものが，1/10秒単位とか1/2秒単位でということで，単位時間を少し粗めに取ることで，許容度を高くしたい。別の記事で解説されていると思うが，入力バッファリングでレイテンシの変動を吸収することも必要だろう。別に通信自体が速くなったりはしないのだが，速度変動による最悪パターンは回避できる。バッファリングすることでリアルタイム性は損なわれるのだが，それをある程度捨てることで得るものも大きいと思われる。

なお，テストプログラムでの実験などで判断すると（このあたりの感覚では個人差は大きそうだが），最大で取った場合，1/4秒周期で回してレイテンシ2というのが限度ではないかと思われる（0.5秒遅れ）。4fpsだと画面的にはきついところもあるが，中割りで見た目だけ整えてやることは不可能ではない。情報量などの転送量は結構確保できるので，そちらを優先するとこんな感じだ。例の「やがて3倍速くなる」仮説でいって，12fpsになったら結構合格点かとも思われる。無論，もっと速いにこしたことはないが，最低で許容されるのはこれくらいだろうと予想している。

なお，これは「これだけは待つけど，遅れたら知らないよ」システムを取ったときの話だ。無制限にハンドシェイクするほど回線を信用してはいないけど，どんどん先に進めていってそれでゲームが成立すると思うほどにも回線を信用していない場合（結局全然信用してないのか）には「ちょっとは待つけど……」ってのは正解だろう。

開始時とか途中の回線状態によって，安定してそうだったらサイクルタイムを速くするとかいう適応型速度対応も考えられなくはないんだけど（不安定だとどんどん遅くする手順のほうが一般的），予備実験をしていないので実際の回線の様子はいまひとつわからない。いろんな環境でデータを取る必要がある。その辺は検討課題だ（周り見ると専用線ばかりだしなあ，ここ……）。

■根本的問題への回帰

ちょっと脱線する。そもそも，なんでそうまでしてネットワークゲームにしなくてはいけないのだろう？

対戦型ゲームの場合，たいていは人間のアタマのほうがCPUより高性能だということが挙げられる。慣れないうちはCPUキャラほど巧みに動かなくても，学習しつつ効率を上げていくのでパターンにはまりにくい。その分多彩な遊び方が楽しめることになる。不確定要素としても働くので，ゲーム展開は無限のバリエーションを持ってくる。同じことをプログラムで行うのはかなり難しいだろう。たとえ不可能でなかったとしても，おそらくコストに見あわないと思われる。

コンピュータキャラクターのアルゴリズムを手抜きできる，というわけではないが，ゲーム性の焦点を，より高性能なパーツに置き換えることができるわけだ。終始対人対戦に頼っているようなゲームデザインは，これはこれでまた問題があるようにも感じられるが，これもまた別の話。

人間の敵は人間。とはいえ，戦って殺し合う（たいていの対戦モノ，DOOM系とかはもろにそうだなあ）だけがネットワークゲームではない。ネットワークRPGの場合は「協調」ということが要求される。「ひとりでは生き残れない」という単純な理由で，ちゃんとコミュニケーションを取り，パーティを組んで行動しないと，いつまでもゲートの前で蛇をツンツンしていなければならない。

協調，というのをシングルプレイヤーのゲームで行うと，結局はゲームデザイナーのシナリオどおりに展開していくというだけの，ある意味わざとらしさがあったのではないかと思う。シングルプレイヤーゲームでは，プレイヤーは主人公であり，多くの場合絶対的正義の化身である。「NPCが勝手に話を進めてっちゃうよ」なゲームも過去には存在したようだが，あまり評判はよくなかったように思う。

多くのマルチプレイヤー型ネットワークゲームでは各プレイヤーは対等の関係である。だからこそ「協調」というものが発生しうる。

最近，周りで流行ってるEverQuestでは体力

値や知力，敏捷性などの基本パラメータはキャラクター生成時のまま変化しない。
「これだと，どんなに頑張っても体力なくてプレートメール着れないんですけど……」
ざっと見ると，確かによほどのキャラでないとプレートのセットは着れそうにない。ま，それはそれでバランスのうちだ。もしかしたらアイテムや魔法などで基礎パラメータを上げることができるかもしれないのだが，固定パラメータというゲームデザインはなかなか興味深いものだ。キャラクターが不完全だからこそ，個性が生まれ，ゲームのやり方にも幅が出てくる。協調も必要になる。
ネットワークゲームの普及ということに関して，PTAの皆様からお墨付きをいただくにはこの手のゲームでなにかお子様の情操教育によさそうなものを誰か作ってくれるのがいちばんいいのだが（どっか作らないかなあ……）。
現代のゲーム機普及による「遊び方」を知らない子供たちには，不足しがちなコミュニケーションを回復するようなものこそが望まれているのかもしれない（つーか，そこのPTA，礼儀と常識を叩き込まないままインターネットの世界にガキを出すんじゃない）。
とにかく協調，あるいは忍耐といった部分をテーマにしうるのもネットワークゲームの特徴かもしれないのだ。ゲーム進行が継続しているものだと，失敗してもリセットが利かない。自分が絶対的な主人公を演じていられるわけでもない。
kazuhisa氏の原稿を見てもわかるように，ゲームによって作られた空間がまさにサイバースペースとしてもうひとつの生活の場として存在しうる，というSFじみた話さえ現実のものとなってきているのだ。今後さらに密着したものにもなりうる。さらにいえば，人生のあり方を変える可能性がある。たかがゲーム，しかしそれくらい大きなものになりうるのだ。

■大規模システムの考察

しかし，話題に上るネットワークゲームが海外産のゲームだけではちょっとさみしい。UOだけがネットワークゲームじゃないぞ，という意見もある。しかし，以下ではやがては打倒UO！　ということでネットワーク型準リアルタイムマルチプレイヤーRPGシステムを中心に考えてみることにする。

●サーバ/クライアントシステム

前提としてゲーム本体をサーバ内で走らせる，回線などは必要なだけは確保できる，それは絶対条件だ。できるだけいい条件が確保できたとして（金さえ出せば解決できる問題だ，たぶん……），そこでなにができるかを考えよう。
UOというのはどういう構成になっているのだろうか。
1台のサーバにログインできるのは最大3000人。これだけで済めばまだいいのだが，NPCの処理などを考えるともっともっと処理はかさむ。ブリタニアの大地がそのまま入っており，人々はそこで生活する。プレイヤーの相互作用などを考えると基本的にゲームはサーバ側で動かしていることになる。いや，かなり凄いことをやってるよなあ。クライアントはもっぱら表示と入力を担当していると考えていいだろう。
最新期待のEverQuestだと，1サーバ1000人程度らしいが，区域ごとに管理がきっちり分かれているので区域別にサーバを分けるのは難しくないだろう。これも利口な方法のように思われる。画面表示などの処理はUOより格段に重く（3D必須），クライアント処理全体が重い。ISDN以上の回線と，高速マシンはほぼ必須という敷居の高いゲームだ。現状ではサーバ規模は小さいものの人口密度も高いので（遠くに行くと死んじゃうのでゲート付近の人口密度が異様に高くなっている），サーバの処理量はかなりのものになるだろう。種族ごとにスタート地が変えてあるので，多少は分散されている可能性はあるが，サーバ数は不明だ。
EverQuestはサーバをスケーラブルに拡張できるアーキテクチャで構成されている可能性が高いので有望そうだが，それにしてもサーバ（管理者）側の設備投資などが非常にかさんでくるのは避けられない。
最近はPCにしてもゲーム機にしてもそれなりの処理能力を持っているので，サーバ側の負荷をどの程度分散できるかを考えてみるのも面白い。もちろん，表示などはクライアントのみで行われる。入出力専用クライアント以外に，なにか処理を分担できるかという問題だ。

■試論：クライアント主導システム

サーバ集中型システムのままゲームシステムの進行自体に必要な処理を分散できるものではないので，最低でもなにか遅延などがあったときでも，そのユーザー単体にしか影響しないものである必要がある。
さらにいえば，サーバ側でのゲーム進行をできうる限り減らす方向性が求められるだろう。
ユーザー同士，ユーザーと設定された世界を結ぶのはサーバの役目なので，それは委譲できないだろうが，ゲーム進行自体はクライアント側に移行できないだろうか，と考える。おそらくはUOにしても，プレイヤーはオブジェクト化されて管理されていると思われるが，クライアントマシン自体をオブジェクトとしてとらえるとサーバの処理はより単純化される。いい方を変えるとユーザーオブジェクトは明確にカプセル化しておき，サーバからのメッセージに対して適切に動作しつつ，サーバ側ではシミュレート行わないということになる。メッセージのやり取りでしか相手の状態を確認できないので，回線が不安定な場合など，たとえばある時点で弾が当たったのに相手が生きているのか死んでいるのかわからない状態になることも考えられる（厳密にいえば弾が当たったのかどうか自体がわからないということだが）。
サーバクライアントシステムで，もっぱらクライアントだけでゲームが動くシステムだともいえるだろう。クライアント同士の相互作用でゲームが構成されるシステムだ。
メッセージのやり取りだけで，それぞれのオブジェクトは勝手に動いていく。ユーザーについてサーバ側はなにも知らなくてもゲームは進行可能だ。NPC処理もクライアント側に移すことができるだろう。たとえば，なにかモノを買ったりするときにサーバが介在する必要はあまりない。金銭とモノの交換が行われておればよいだけなので，処理が終わったときには，ユーザーの持ち物の構成が変化しているだけ。そしてユーザーの持ち物についてサーバが把握している必要はないし，買い物が行われたという事実すらサーバには不要なものだ（売ったものがずっと店に残っていたりする場合は別として）。
ユーザーパラメータについては感知しない。
なお，このようなシステムの欠点は，データハッキングが容易だということだ。ユーザー側にデータがあるのでサーバの知らないところでデータが改竄されても，サーバは最初から感知してないのだからわからない。まあ，チェック機構はつけられそうだが，本質部分は変わらない。サーバは世界を構成するオブジェクトをすべて内蔵はするものの，処理の基本はオブジェクト間のメッセージを転送するだけ。最小限のサーバコストで最大限のことを行うことができるのは，やはりこのようなシステムになるはずだ。サーバの抱えるオブジェクトへの処理だけでも膨大という説はあるのだが，それはしかたない（さすがに分散処理は危険だ）。
一般のネットワークゲームを見ていると，通信エラー処理をそれなりにやっていても，たまに2台で別のゲームが動くことがある。完全にサーバ管理型のものならともかく，これは避けがたいように思われる。
エラーが発生したときのことを考えてみよう。データの受け渡し全部をチェックするのは無駄なので，エラーの影響がその場限りで波及しないようなシステムが望ましい。エラーが起きても破綻しないように組んでいくわけだ。
でないと，壁にぶつかったまま走っている人を見て不審に思ったり，片方では死んでいる人がもう片方では生きていたりすることもありうるかもしれない。これはピアトゥピアで並列して同じゲームが走っていて，相互の情報が違うときに発生する。これは上手く処理しないと致命的だ。

■森の木

アマゾンでもシベリアでもどこでもいいのだが，原生林の真ん中にある大木が倒れたとしよう。大きな木なので倒れれば地響きだってするだろうし，鳥の巣は壊れ，下敷きになって虫などがつぶれて死んでしまったかもしれない。木の真下辺りでは生死にかかわる問題だが，少し離れるとその影響はだんだん薄くなる。1kmも離れればほとんどなにもなかったかのごとく，木の倒れた影響は拡散してしまっているだろう。それが何100kmも離れたところにいるあなたになにか影響してい

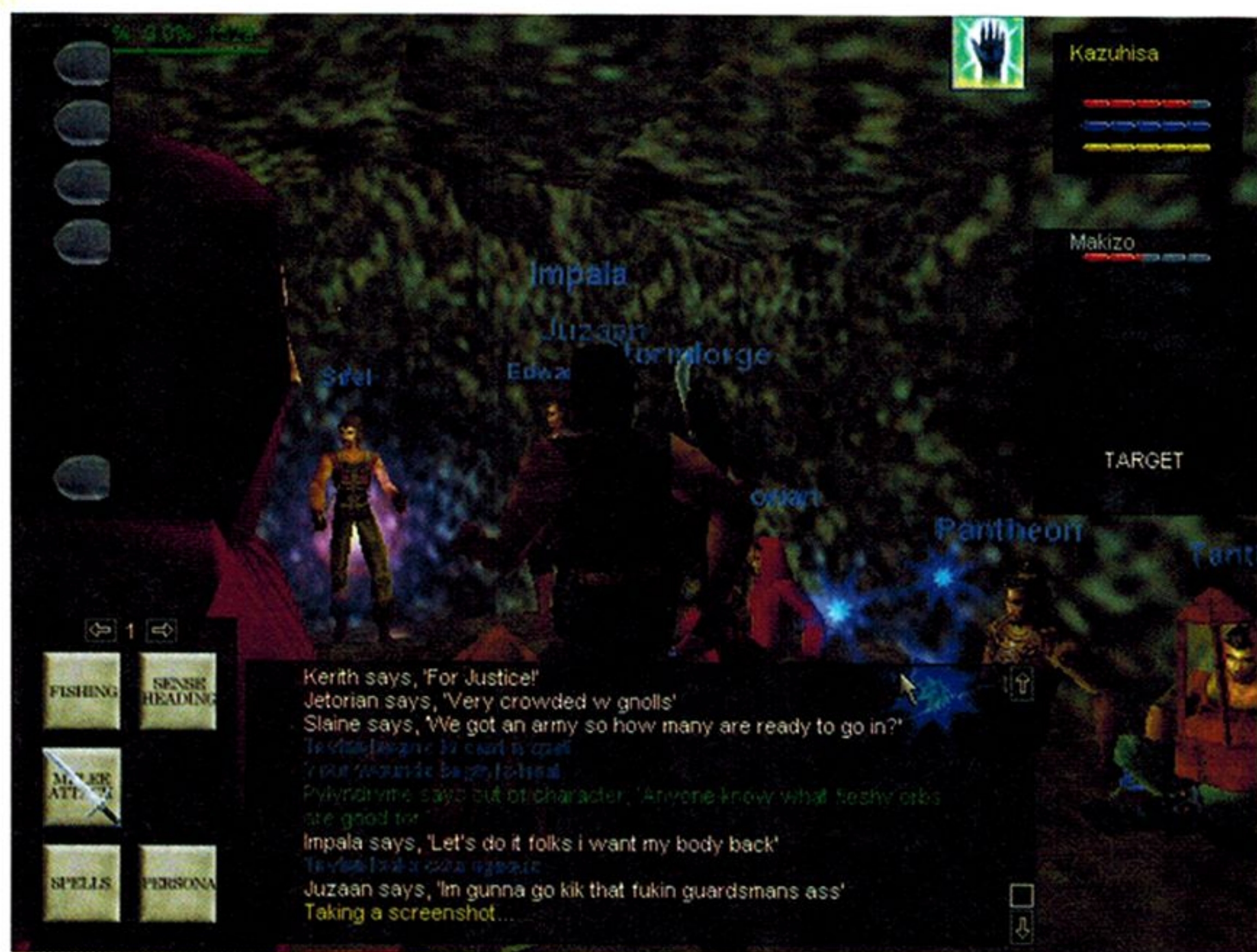



図1　これはEverQuest。Gnollの群れに追われてエリア境界まで逃げ延びた面々。「どこの馬鹿があんなの連れてきたんだ！」「私の死体取りに行かなきゃならないんですけど」「討って出れる奴は何人いる？」……安全地帯では自然にパーティが発生する。これが"デバッグ"されたら大変なことになりそうな

るものはあるだろうか。なにも違いが検出されない場合，その木が倒れたという現象はあなたにとって本当に存在していた現象なのだろうか？　というよく知られた命題がある。

答えはない。

巡り巡ってなんらかの影響を与えているとする説，たとえば今朝食べた味噌汁が少ししょっぱかったことにもこの木が倒れた影響が微妙に関与していたはずだとする人もいるだろう。

もう一方で，存在していようといまいと関係がないとする説もある。この命題に対してはこちらのほうが一般的だ。なにが起こっていようとある人にとって認知できる影響がなにひとつなかった場合には，その事象自体がその人にとっては存在していないとする説だ。認知されて初めて存在は意味を持つものであり，認知不可能な事象は存在していてもしていなくても変わりはない。もっと極端にいえば，認知されるまでは事象は存在していないのと同じなのだ，とする説だ。

さっきの続きで，UOよりも広大なマップと無制限のキャラクターを扱うゲームを想定してみよう。こうなるとすべてのキャラクターについての管理を一括して行うことは難しい。

EverQuestなどはUOよりは小さなスケールだが，マス目ごとにエリア分けされている。境界があるというのも興醒めなのでサーバ間はシームレスに扱いたい（現状では，サーバの境界を越えて敵は追ってこない。バグという話だが，困っている人はいないかも。図1参照）。

これだとデータの食い違いがさけられないのではないか，というのが先ほどまでの結論だった。では，ここからは「食い違っていてマズイのか」という話に入ろう。

大陸の反対側では戦争が起こっているかもしれない。が，そのメッセージが届くまではそのオブジェクトに影響が表れる必要はない。情報が入ってから初めて存在させても十分に間にあう話だ。

大陸の端にいる別キャラクターのパラメータの変動などは，逆端にいるキャラクターに影響しない（普通は）。逆にそんな些細なものでも影響を出させることもできる。メッセージを送ればいいだけだ。

メッセージを受けとったオブジェクトがどのように反応するかは定かではない。ほかのオブジェクトに同じメッセージを転送するか，特定の処理を行うか，あるいは単に無視する。ほら，なんとなくイベントループの処理コードが見えてきた人もいるだろう。

すべてのオブジェクトをイベントドリブンに作って作った場合，どのようなことが起こるだろうか。また，メッセージのやり取りにエラーがあるとどうなるだろう。

■矛盾は出るか

リンゴの木があったとしよう。実がなっている。2人の人が遠くから弓矢で同時に狙った。メッセージの伝播速度は一定ではないとすると，それぞれで矛盾は生じないだろうか？

リンゴは先にやってきたメッセージに対してアクションを返す。次にやってきたメッセージに対しては時系列に従って適切なメッセージを出せばいい。

「はずれ」

一般の物理現象ではピアトゥピアの現象認識がされるので，観測者の視点から見たものというものが絶対視されるが，ここでは物体自体が自分の状態を知っている。

不確定性原理も相対性原理も「観察」というものを絶対視することから発生したようなものだが，このような世界では事象は実にシンプルに処理できる。観測者によって流れる時間が一定でなくても，結果はひとつに確定できる。受けとるエラーレートなどで情報が確率でしか表されなくても，出てきた結果は確定的だ。もちろん，メッセージをやり取りすること自体でオブジェクトの状態が変化する状況をわざわざ作らない限りは，だが。

認識者の視点から見た処理がそのユーザーと少なくともその周辺の十分な範囲に及べば，すくなくとも膨大なユーザーを管理しなくても済みそうに思われる。

情報伝達部分でいくつかマズそうな点はあるのだが，今後のゲームの作り方ってのはどのみちこんな感じになるはずだ。物理系のシミュレータも当然組み込まれる。デバッグは至難だろうし，プログラムした側にもたぶんなにが起こるのかわからないだろうなあ。こういうのってどんなものだろうか。

■コンシューマ機の憂鬱

パソコンではほぼ無制限といえるハードディスク容量（最近では100MBくらいなら使っても怒られないだろう），潤沢なメモリというハードウェア環境を想定してさまざまな処理ができる。

DreamcastやPS2といったコンシューマゲーム機をはじめ，パソコンの性能向上などまで考えると，クライアントマシンの性能はどんどん上がっていくことは明白だ。サーバの性能も上がるだろうが，それは接続数を増やすか処理量を増やすかの選択でしかないので，本質的な部分での進歩ではない。ネットワークの普及とともに1クライアント当たりに割けるサーバ能力は減ってくる可能性すらある。ネットワークの普及速度よりもサーバコストパフォーマンスの向上度が大きくないとユーザーのメリットはない。回線が桁違いに速くなれば別なのだが。しかし，今後，回線の品質が大幅に向上することは当分ないようにも思われる。

コンシューマ機の場合「強力なクライアントCPU性能」ただし，「ディスクレスでメモリも少ない」……という特殊な環境が構築される。サーバ側に期待できないならクライアント側で頑張ってみようという考え方で話を進めていたのだが，コンシューマ機にそのままは通用しそうにない。

そうなると，ひたすら画面や音声を派手にしていくという方法しかないのだろうか。せっかくのCPU性能なのだけどなあ。

クライアントが入出力端末用途に限定されると，回線は一定だし，ゲーム自体の規模はサーバで決定されるようになってくる。ネットワークゲームでは，革新的なサーバハードウェア技術，あるいは並列ソフトウェア技術などが今後の焦点になっていくのだろうか。

特集 ネットワークゲームの地平へ

多体問題と相互作用を考える

三沢和彦 *Misawa Kazuhiko*

ネットワークゲームに限ったことではないのですが，マルチプレイヤーゲームでは，それぞれのプレイヤーの相互作用などをうまく処理しなければいけません。しかし，展開に3つ以上のオブジェクトが絡んでくると，ちゃんと解くことが非常に難しくなってくることがあります。ここではそういった多体問題の解決法を見てみましょう。

■ゲームとの類似性

私のお気に入りのゲームはセガラリー2である。選ぶクルマはスバルWRCに限る。というのも，私の愛車はスバルのインプレッサWRX，サーキットライセンスまで一応取得するぐらいのラリーファンだからである。だから，Dreamcastでやるよりは，アーケード版のほうが臨場感があっていい。

ところで，レースゲームで自車と敵車と衝突したときの処理というのはどうしているのだろうか，と不思議に思う。自車と敵車の位置や速度のデータを持っていて，衝突した瞬間に弾かれる方向や勢いを計算しているのだろう。また，衝突したときに受ける衝撃やステアリングの手応えも，自車と敵車の位置や速度の関係でその都度違うだろうから，計算はさぞかし大変であろう。

プレーしている人間に気づかれないようなかたちで，どこかで，うまく計算を簡便化してやる必要がある。実際問題として，対戦型ゲームで敵と衝突したときの処理を物理的に厳密に行うのは難しいと思われる。

筆者が専門としている物理学の世界では，このように衝突したり，引きつけあったりして相互作用しあっている複数の物体の挙動を求めることを，多体問題と呼んでいる。

■多体問題とは

多体問題とは，そもそも物理学あるいは天文学で頻繁に使われる用語である。

自然界の存在する物質はすべて電子と原子核で構成されている。これらの物質の性質は物理法則に基づいた基本方程式を解くことによって解明できる。しかし，物質は数え切れないほどの電子で構成されており，その1つひとつが相互作用を及ぼしあっている。その結果，基本方程式の正確な解を求めるのは現実的には不可能といってよい。しかしながら，厳密な解は不可能としても，ある程度の近似を行い，また数値計算の方法を工夫するなどにより，可能な限り正確に解こうという努力が続けられてきた。このように，多数の物体相互の運動を調べる問題を「多体問題」という。

実際の物理現象の解析では，着目する1体の運動に対して，その他すべての物体の影響をならして考える方法がとられてきた。周りの物体を1個1個別々に追跡するのは，労力の割には意味がないことも多い。このように，多体問題を，対象とする1体と周囲の環境とに切り離して考えるやり方を「平均場近似」という。平均場近似では問題となる1体以外の影響を，その1体にとっての周辺環境だとみなすところにポイントがある。この平均場近似を用いて，多くの物理現象の本質的な部分は解決されてきたといっても過言ではない。この平均場近似を出発点として，より詳細に物体間の相互作用の影響を取り込んだ解法が見出されてきたのである。

複雑に絡みあう物体間の運動の様子をいかに簡便に効率よく，いかにより厳密に解くかは，物理学者の数学的知識，計算機的素養，そして物理学的センスによるところが大きい。センスがなければ，実際の物質における多体問題の本質を明らかにすることはできない。

このような，物理学における多体問題の事情は，ゲームプログラミングにも，あてはまるものであろう。

物理学，化学および天文学の分野では，このような多体問題を解くということが一般的な命題である。そのために，それぞれの分野で，多体問題を計算することだけに特化した計算機が開発されている。

したがって，ゲーム専用のアーキテクチャを持ったRISCというものが開発されてもおかしくないのかもしれない。

■2体問題の厳密解

まず，相互作用する物体が2個しかない場合を考えてみよう。(物理学の計算としては)最も簡単でかつ身近な例を取り上げる。それは，地球と月の運動である。2体問題であるから，地球と月しか考えない。地球と月の間はニュートンが発見した万有引力が働いている。この万有引力というのは，2つの物体の間で互いに引き合う力であり，その力の大きさは2体間の距離の2乗に反比例する。これはつまり，2物体が遠くにあればあるほど，互いに引き合う力は弱くなり，近くにあればあるほど力は強くなるということである。

■物体の運動を記述する物理法則

物体の運動を決める物理量には，位置と速度がある。この2つの量がそれぞれの物体についてわかれば，全体の運動を追跡することができる。ここで，位置と速度はともに時間の関数である。つまり，ある時刻における位置と速度の推移を計算できればよい。

ところで，「位置」と「速度」とはなんだろう。「位置」というのは，空間上のある点のことである。いま我々の住んでいる空間は3次元空間であり，そこでの位置は，x，y，zという独立な3方向成分で表される。だから，3つの量で位置は表せる。また，速度というのは「単位時間当たりの位置の変化分」と言葉ではいい表わすことができるが，実際は微分という概念が必要になってくる。

物体の運動を記述するための基本法則は，ニュートンの運動方程式と呼ばれるものである。この基本法則は，加速度が力に比例する，というものである。加速度というのは「単位時間当たりの速度の変化分」のことであり，時間とともに速度がどのように変化していくかを表す量である。これまた，微分の概念からすると，速度の時間微分となる。加速度と力の比例係数を物体の質量という。

この運動方程式のみを使って，万有引力を及ぼしあう地球と月の運動を導いてみよう。

■ハミルトニアン

ニュートンの運動方程式は，物体の質量m，速度の時間微分 dv/dt，物体の受けた力Fの間に成り立つ以下の関係式である。

$$m\frac{dv}{dt}=\mathrm{F}$$

いま，物体が地球と月の2つあるから，それぞれの物体について別々に運動方程式が立てられる。この時，地球が月から受ける力は月が地球から受ける力と同じ大きさで，向きが逆である。これは，作用と反作用の法則といわれるもので，たとえば，われわれが手で壁を押すときに，手が壁を押すのと同時に，壁から押し戻されている感じがするのと同じである。

F　-F

力をベクトル量で表すと，向きはプラス/マイナスの符号で表せる。上の図のように，地球が月に引かれる力をFとおくと，月が地球に引かれる力は逆符号の力になるので，－Fということになる。このFおよび－Fの力を使って，運動方程式を立てると，

$$m_1\frac{d\mathbf{v}_1}{dt}=\mathbf{F}$$
$$m_2\frac{d\mathbf{v}_2}{dt}=-\mathbf{F}$$

となる。ここで，1は地球，2は月を表す添字である。

ニュートンの万有引力の法則より，2物体間にはたらく引力は2体間の距離の二乗に反比例する。これを式で書くと，

$$F=\frac{-Gm_1m_2}{r^2}\quad \mathbf{F}=\frac{-Gm_1m_2}{|\mathbf{r}_1-\mathbf{r}_2|^2}\frac{\mathbf{r}_1-\mathbf{r}_2}{|\mathbf{r}_1-\mathbf{r}_2|}$$

のようになる。どうして，このような式で書けるのか深く考え出すときりがない。というよりは，むしろケプラーが太陽系の惑星の運行を観測していた結果をニュートンが丹念に考察したところ，たまたまこのような式で2体間に働く力を考えれば，うまく説明できると気がついただけにすぎない。ここで，2体間の距離というのは，

$$\mathbf{r}\equiv\mathbf{r}_1-\mathbf{r}_2$$

で表されるから，力は$\mathbf{r}\equiv\mathbf{r}_1-\mathbf{r}_2$の関数で書き表せる。

■重心系

次に，重心という概念を考えてみよう。重心座標R次のように定義する。

$$\mathbf{R}\equiv\frac{m_1\mathbf{r}_1+m_2\mathbf{r}_2}{(m_1+m_2)}=\frac{m_1\mathbf{r}_1+m_2\mathbf{r}_2}{M}$$
$$\mathbf{r}_1=\mathbf{R}+\frac{m_2}{(m_1+m_2)}\mathbf{r}$$
$$\mathbf{r}_2=\mathbf{R}-\frac{m_1}{(m_1+m_2)}\mathbf{r}$$
$$M\equiv m_1+m_2$$

この重心座標を用いて最初の運動方程式を書き換えてみよう。

$$m_1\frac{d\mathbf{v}_1}{dt}+m_2\frac{d\mathbf{v}_2}{dt}=0$$
$$m_1\frac{d^2\mathbf{r}_1}{dt^2}+m_2\frac{d^2\mathbf{r}_2}{dt^2}=M\frac{d^2\mathbf{R}}{dt^2}=M\frac{d\mathbf{V}}{dt}=0$$

最後の式は，地球と月とを2つあわせて，全体をひとつの物体だとみなしたときに，その全体が満たす運動の法則を示す方程式になっている。このように，複数の物体の全体をひとつとして考えたときに，その全体の質量が集中していると考えられる点を重心という。重心の方程式を見ると，地球と月とをあわせた全体には，外部から力が働いていないということがわかる。したがって，地球と月の重心は停止しているとして取り扱ってよい。すなわち，

$$\mathbf{R}=0$$

とおいてよい。

■相対運動

今度は，地球から見た月の相対的な運動を考えてみよう。相対的な運動というのは，2体間の距離$\mathbf{r}\equiv\mathbf{r}_1-\mathbf{r}_2$が時間的に推移していく様子を調べることに対応する。

$$\mathbf{r}_1=\frac{m_2}{(m_1+m_2)}\mathbf{r}$$
$$\mathbf{r}_2=-\frac{m_1}{(m_1+m_2)}\mathbf{r}$$
$$\mathbf{R}\equiv\frac{m_1\mathbf{r}_1+m_2\mathbf{r}_2}{(m_1+m_2)}=\frac{m_1\mathbf{r}_1+m_2\mathbf{r}_2}{M}$$
$$\mathbf{r}_1=\mathbf{R}+\frac{m_2}{(m_1+m_2)}\mathbf{r}$$
$$\mathbf{r}_2=\mathbf{R}-\frac{m_1}{(m_1+m_2)}\mathbf{r}$$
$$M\equiv m_1+m_2$$

という定義式の両辺を時間微分すると，

$$\mathbf{v}_1=\frac{m_2}{(m_1+m_2)}\mathbf{v}$$
$$\mathbf{v}_2=-\frac{m_1}{(m_1+m_2)}\mathbf{v}$$

と書けるが，この式をもともとの運動方程式に代入すると，

$$m_1\frac{d\mathbf{v}_1}{dt}=\mathbf{F}$$
$$m_2\frac{d\mathbf{v}_2}{dt}=-\mathbf{F}$$

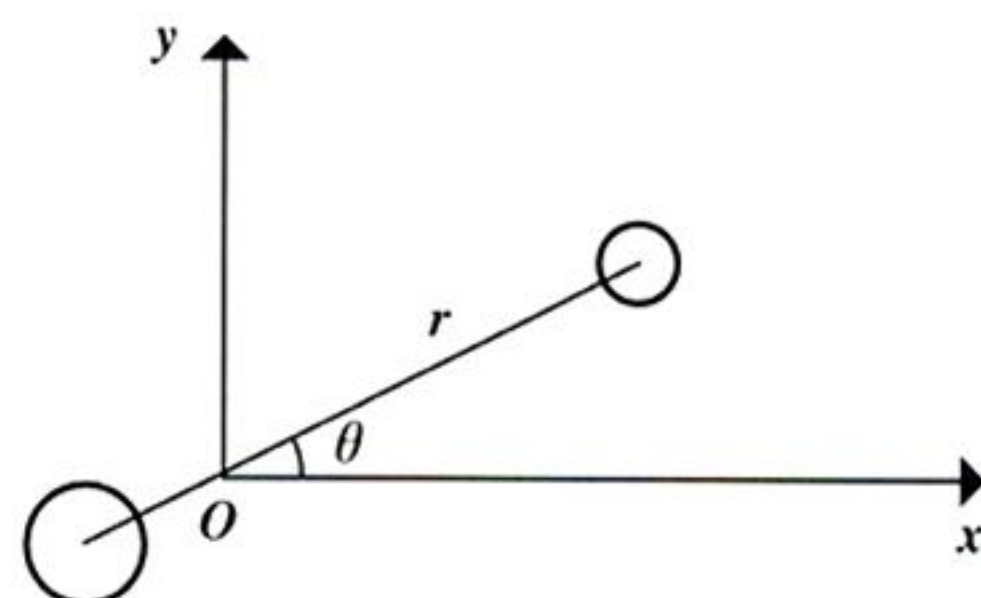
最終的に得られた方程式は，

$$\mu\frac{d\mathbf{v}}{dt}=\frac{-Gm_1m_2}{|\mathbf{r}|^2}\frac{\mathbf{r}}{|\mathbf{r}|}$$
$$\mu\equiv\frac{m_1m_2}{(m_1+m_2)}$$

となる。ここで定義したμは，換算質量と呼ばれるもので，地球と月の相対的な運動は換算質量を持つ仮想的な物体の運動に置き換えて解くことができる。

■極座標

このような2物体間の運動を解くには，極座標という物体の位置の表し方が便利である。この極座標というのは，物体間の距離と方位角で位置を表すやり方である。地球を周回する月は，常に地球の周りを離れずにぐるぐる回っているだけであるから，距離と方位角で示すのが，もっともわかりやすいからである。

y, r, θ, O, x

極座標は，普通一般に用いられる「直交座標」系と以下のような関係にある。

$$x=r\cos\theta$$
$$y=r\sin\theta$$
$$r^2=x^2+y^2$$

このように，異なる座標の表し方の間の関係式を，座標変換(式)と呼ぶ。さらに，物体の運動方程式では位置だけでなく，速度も計算に用いるので，速度の変換式も必要になってくる。しかしながら，速度の変換式を導出する過程を理解するには，ある程度の物理学・数学的素養が要求されるので，ここでは単に書き下すだけに留めておく。

$$r\frac{dr}{dt}=x\frac{dx}{dt}+y\frac{dy}{dt}$$
$$\frac{dx}{dt}=\frac{dr}{dt}\cos\theta-r\frac{d\theta}{dt}\sin\theta$$
$$\frac{dy}{dt}=\frac{dr}{dt}\sin\theta+r\frac{d\theta}{dt}\cos\theta$$
$$r^2\frac{d\theta}{dt}=x\frac{dy}{dt}-y\frac{dx}{dt}$$

■角運動量保存の法則

以上で，地球を周回する月の軌道を計算する準備は整った。では，実際に月の運動を導いてみよう。周回する物体の運動を理解するのに，角運動量という概念は非常に重要である。角運動量というのは，ある点を中心にどのくらいの距離をどのくらいの速度で回転するかを表す量である。単位時間当たりの回転角の変化分と距離の二乗を掛けあわせたものが角運動量に比例する。

$$I(t)=\mu r^2\left(\frac{d\theta}{dt}\right)$$
$$\frac{1}{\mu}\frac{dI(t)}{dt}=\frac{d}{dt}\left\{r^2\left(\frac{d\theta}{dt}\right)\right\}=\frac{d}{dt}\left(x\frac{dy}{dt}-y\frac{dx}{dt}\right)$$
$$=x\frac{d^2y}{dt^2}-y\frac{d^2x}{dt^2}=0$$

ここで，角運動量の計算には，前節の速度の変換式を用いた。最後の行で式の値が0になる事情は相対運動の運動方程式からわかる。

$$\frac{d\mathbf{v}}{dt}=x\frac{d^2\mathbf{r}}{dt^2}=\frac{-Gm_1m_2}{\mu r^3}\mathbf{r}$$
$$\frac{d^2x}{dt^2}=\frac{-Gm_1m_2}{\mu r^3}x$$
$$\frac{d^2y}{dt^2}=\frac{-Gm_1m_2}{\mu r^3}y$$
$$x\frac{d^2y}{dt^2}-y\frac{d^2x}{dt^2}=0$$

角運動量の時間微分 $\frac{dI(t)}{dt}$ が0になるということは，角運動量の値は時間とともに変化しないということになる。これを角運動量保存の法則という。角運動量保存の法則は，いま考えているように，物体に働く力が回転の中心に向かっているときに成り立つ。実際，地球と月の間の引力は地球と月を結ぶ直線に沿って働いており，かつ，回転運動の中心もその直線上にある。

$$r^2\left(\frac{d\theta}{dt}\right)=\frac{I}{\mu}=\text{const.}$$

■エネルギー保存の法則

惑星の運動でもっとも重要な法則のひとつである，角運動量保存の法則を導いた。次にもうひとつの大切な法則について考えてみよう。それはエネルギー保存の法則である。物体の運動エネルギーを次のように定義する。

$$U=\frac{m}{2}v^2$$

速度が速いほど，運動の勢いが強いので，エネルギーを多く持つと考えればよい。もちろん，同じ速度で動いていれば，質量の大きいものほど勢いが強いのもわかるだろう。これもまた，角運動量保存の法則と同じように，エネルギーの時間微分を計算する。

$$\frac{m}{2}\frac{dv^2}{dt}=mv\cdot\frac{dv}{dt}=F\cdot\frac{dr}{dt}$$
$$\frac{m}{2}\frac{dv_1^2}{dt}=m_1v_1\cdot\frac{dv_1}{dt}=F_1\cdot\frac{dr_1}{dt}$$
$$\frac{m_2}{2}\frac{dv_2^2}{dt}=m_2v_2\cdot\frac{dv_2}{dt}=F_2\cdot\frac{dr_2}{dt}$$

運動方程式を変形した。また，地球と月の間に働く力は互いに逆向きで同じ大きさだから，

$$F_1=-F_2=F=\frac{-Gm_1m_2}{|r_1-r_2|^2}\frac{r_1-r_2}{|r_1-r_2|}$$
$$r\equiv r_1-r_2$$
$$\frac{m_1}{2}\frac{dv_1^2}{dt}+\frac{m_2}{2}\frac{dv_2^2}{dt}=\frac{-Gm_1m_2}{|r|^2}\frac{r}{|r|}\cdot\frac{dr}{dt}=\frac{d}{dt}\frac{Gm_1m}{|r|}$$
$$\frac{d}{dt}\left(\frac{m_2}{2}v_1^2+\frac{m_2}{2}v_2^2-\frac{Gm_1m_2}{|r|}\right)=0$$

$\frac{-Gm_1m_2}{|r|^2}\frac{r}{|r|}\cdot\frac{dr}{dt}=\frac{d}{dt}\frac{Gm_1m_2}{|r|}$ の部分は，かなり難しいと思うので，読み飛ばしてほしい。結局，

$$\frac{m_2}{2}v_1^2+\frac{m_2}{2}v_2^2-\frac{Gm_1m_2}{|r|}=E=\text{const.}$$

となる。左辺の第1項 $\frac{m_2}{2}v_1^2$ と第2項 $\frac{m_2}{2}v_2^2$ とは，それぞれ地球と月の運動エネルギーである。第3項 $-\frac{Gm_1m_2}{|r|}$ は，2体間の距離とそれぞれの質量で決まる量で，これを位置エネルギーと呼んでいる。エネルギー保存の法則はこれらのエネルギーの総和が常に一定である，ということをいっている。この式をまた極座標の座標変換式で変形すると，

$$\frac{\mu}{2}\left|\frac{dr}{dt}\right|^2-\frac{Gm_1m_2}{|r|}=E$$
$$\left(\frac{dx}{dt}\right)^2+\left(\frac{dy}{dt}\right)^2=\left(\frac{dr}{dt}\right)^2+r^2\left(\frac{d\theta}{dt}\right)^2$$
$$\frac{\mu}{2}\left(\frac{dr}{dt}\right)^2+\frac{\mu}{2}r^2\left(\frac{d\theta}{dt}\right)^2-\frac{\gamma}{r}=E$$
$$\gamma=Gm_1m_2$$

さらに，角運動量保存の法則を用いると，

$$\frac{\mu}{2}\left(\frac{dr}{dt}\right)^2+\frac{I^2}{2\mu r^2}-\frac{\gamma}{r}=E$$

という簡単な式になる。これまで，フォローするのにかなり苦痛な式を書き連ねてきたが，これでも，地球と月のたった2体間で起こる力のやり取りしか扱っていない。しかもこの2体は宇宙空間を運動していて，よそからはなんの力も受けていない。いかに物理的に厳密に解くのが困難かわかるだろう。

■ケプラー問題

この最後の式をさらに変形していくと，実際の月の軌道が求められる。それには，

- 双曲線
- 放物線
- 楕円
- 円

などの軌道がある。

$$\frac{d\theta}{dt}=\frac{I}{\mu r^2}$$
$$\frac{\mu}{2}\left(\frac{dr}{dt}\right)^2+\frac{I^2}{2\mu r^2}-\frac{\gamma}{r}=E$$

これらの式を変形して，

$$\frac{dr}{\sqrt{\frac{2}{\mu}E+\frac{2\gamma}{\mu r}-\frac{I^2}{\mu^2r^2}}}=dt=\frac{\mu r^2}{I}d\theta$$
$$d\theta=\int\frac{I\,dr}{\mu r\sqrt{2\mu r^2E+2\mu r\gamma-I^2}}$$

この積分も数学の知識があると解くことができて，

$$r=\frac{\lambda(1+\varepsilon)}{1+\varepsilon\cos\theta}$$
$$\varepsilon\equiv\sqrt{1+\frac{2EI^2}{\mu r^2}}$$
$$\lambda\equiv\frac{I^2}{\mu r^2}\cdot\frac{1}{1+\varepsilon}$$

と結構簡単な式になる。

■シミュレーション

この軌道を実際に計算してみよう。ここで重要なパラメータは ε である。ε の値によって軌道の形が決まるのである。そこで，ε をいくつか変えて計算した結果を示す。一方，λ については，$\cos\theta$ の最大値が1であるということを考慮する。

$$\cos\theta\leq 1$$
$$\frac{1+\varepsilon}{1+\varepsilon\cos\theta}\geq\frac{1+\varepsilon}{1+\varepsilon}=1$$

■ Keplar problem

```
In[1]:=  Clear[r]

In[2]:=  r[ε_, λ_] := λ (1 + ε) / (1 + ε Cos[θ])

In[3]:=  x[θ_] := r[ ε, λ] Cos[θ];
         y[θ_] := r [ε, λ] Sin[θ];
```

■ Simulation

■ Circle

```
In[4]:=  ε = 0;
         λ = 1;

In[5]:=  circle = ParametricPlot[{x[θ], y[θ] }, {θ, 0, 2 Pi},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];

         General::spell1 :
          Possible spelling error: new symbol name "circle" is similar to existing symbol "Circle".
```

■ ellipsoid

```
In[6]:=  ε = 0.5;
         λ = 1;

In[7]:=  ellipsoid = ParametricPlot[{x[θ], y[θ] }, {θ, 0, 2 Pi},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];
```

■ parabola

```
In[8]:=  ε = 1;
         λ = 1;

In[9]:=  parabola = ParametricPlot[{x[θ], y[θ] }, {θ, -2.2, 2.2},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];
```

■ hyperbola

```
In[10]:=  ε = 2;
          λ = 1;

In[11]:=  hyperbola = ParametricPlot[{x[θ], y[θ] }, {θ, -1.8, 1.8},
          PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];
```

図1 Mathmaticaでのケプラー問題処理過程

$$r = \frac{\lambda(1+\varepsilon)}{1+\varepsilon\cos\theta} \geq \lambda$$

計算によって，rの最小値がλであることがわかる。rに最小値があるということは，月と地球はある距離以上は近づくことはないということである。したがって，地球と月は衝突しないですむ。

また，一方，$0 \leq \varepsilon < 1$ の場合は，

$$\cos\theta \geq -1$$

$$\frac{1+\varepsilon}{1+\varepsilon\cos\theta} \leq \frac{1+\varepsilon}{1-\varepsilon}$$

$$r = \frac{\lambda(1+\varepsilon)}{1+\varepsilon\cos\theta} \leq \lambda\frac{1+\varepsilon}{1-\varepsilon}$$

となって，距離の最大値も存在する。距離の最大値があるということは，月はどこまでも遠くに離れていかないということである。

以下のシミュレーションでは，$\lambda = 1$ に固定して計算した。

計算は数値解析ソフトの最高峰といえるWolfram Research社のMathematicaを用いた。

もっとも特殊な例は，$\varepsilon = 0$ の場合である。この場合は，方位角θによらず，距離rはλで一定である。この一定値を $r = \lambda = r_0$ と置くと，

$$x = r_0\cos\theta$$
$$y = r_0\sin\theta$$
$$r_0^2 = x^2 + y^2$$

となり，この場合は円軌道となることがわかる。

$0 < \varepsilon < 1$ では，もっとも近い距離ともっとも遠い距離の間を近づいたり遠ざかったりしながらぐるぐる回るが，このときの軌道の形は円を一方向につぶした楕円の形になっている。シミュレーションでは，$\varepsilon = \frac{1}{2}$ の場合を示した。

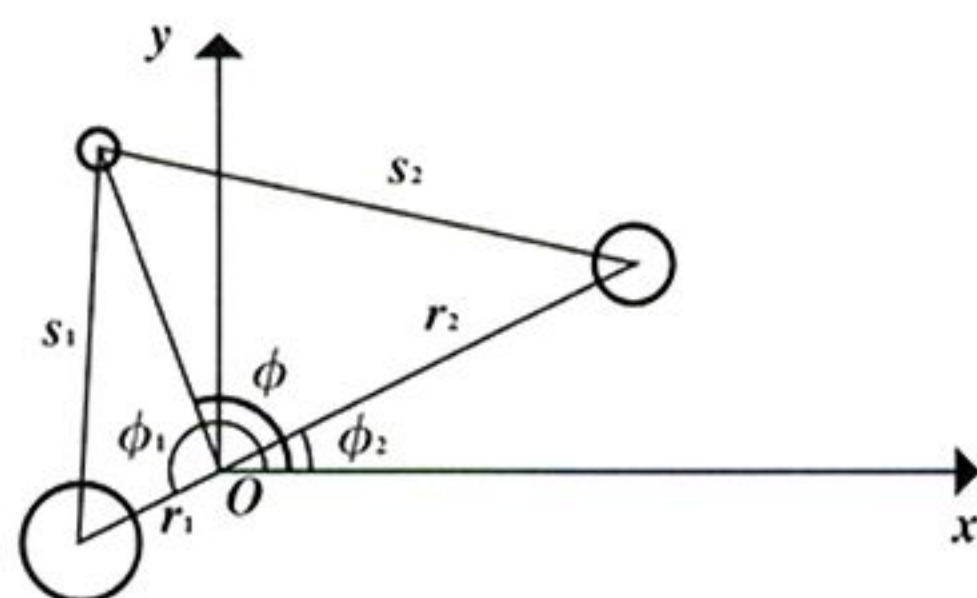
このように，地球を周回する月の軌道の問題は，きわめて数学的に美しい幾何曲線の解を得ることができた。

■アポロ宇宙船，帰還せよ！ 制限3体問題

地球を周回する月の運動は，月が無限遠に飛んでいかない限り，円軌道か楕円軌道であることがわかった。次に今回の本題である，地球から打ち上げられた宇宙船が月を巡って，地球に帰還するケースを考えよう。物体は，地球，月と宇宙船の3体になる。ここでは，上で述べた2体問題を3体に拡張すればよいと思われるが，話はそんなに簡単には終わらない。2体では解けた問題が，3体になると，途端に厳密には解けなくなってしまうのである。それでもなんとかして解を得るには，いろいろな近似を工夫しなければならない。今回考えるアポロ宇宙船のケースでは，次のような近似を取り入れることにする。

(1)地球を周回する月の軌道は円軌道とする
(2)宇宙船の質量は地球と月の質量に比べて無視できる。これはつまり，宇宙船の動きによって地球と月の運動が影響を受けないということである
(3)地球と月と宇宙船は，同一平面上を運動する

このような近似を取り入れた3体問題を制限3体問題と呼んでいる。実際，宇宙船や人工衛星の軌道を計算するには，この方法が一般的である。

ここでは，図で示すような，物体系を考える。

■宇宙船の運動方程式

宇宙船の運動方程式からエネルギー保存の法則を書き下してみよう。宇宙船の質量をmとして，

$$\frac{m}{2}\left(\frac{dr}{dt}\right)^2 + \frac{l^2}{2mr^2} - \frac{Gmn_1}{s_1} - \frac{Gmn_2}{s_2} = E$$

となる。これまでの2体問題と異なる点は，宇宙船が，地球と月と両方から力を受けるという点である。したがって，物体間の距離と質量に関係する位置エネルギーは地球と月と両方からの寄与を加えなければならない。

$$s_1 = \sqrt{r^2 + r_1^2 - 2rr_1\cos(\phi - \phi_1)}$$
$$s_2 = \sqrt{r^2 + r_2^2 - 2rr_2\cos(\phi - \phi_2)}$$

となる。このように，位置エネルギーに現れる物体間の距離が2体の場合とは比較にならないほど複雑になる。これが，3体問題が一般的には解けない理由である。

■シミュレーション

宇宙船の運動もMathematicaでシミュレーションしてみた。地球からある方位，ある速度で打ち上げられた宇宙船が月の近くに向かい，月を巡って地球に戻ってくる行程が実際に得られるか計算してみよう。

(ケース1)地球から脱出できない場合

地球から脱出できないのは，地球から打ち上げられたときに地球の重力に打ち勝つことができない場合である。シミュレーションでは打ち上げの速度を極端に小さい値にしてみた。地球の周辺をぐるぐる回っているだけで，月のほうには向かっていかないことがわかる。

図2 算出された月の軌道

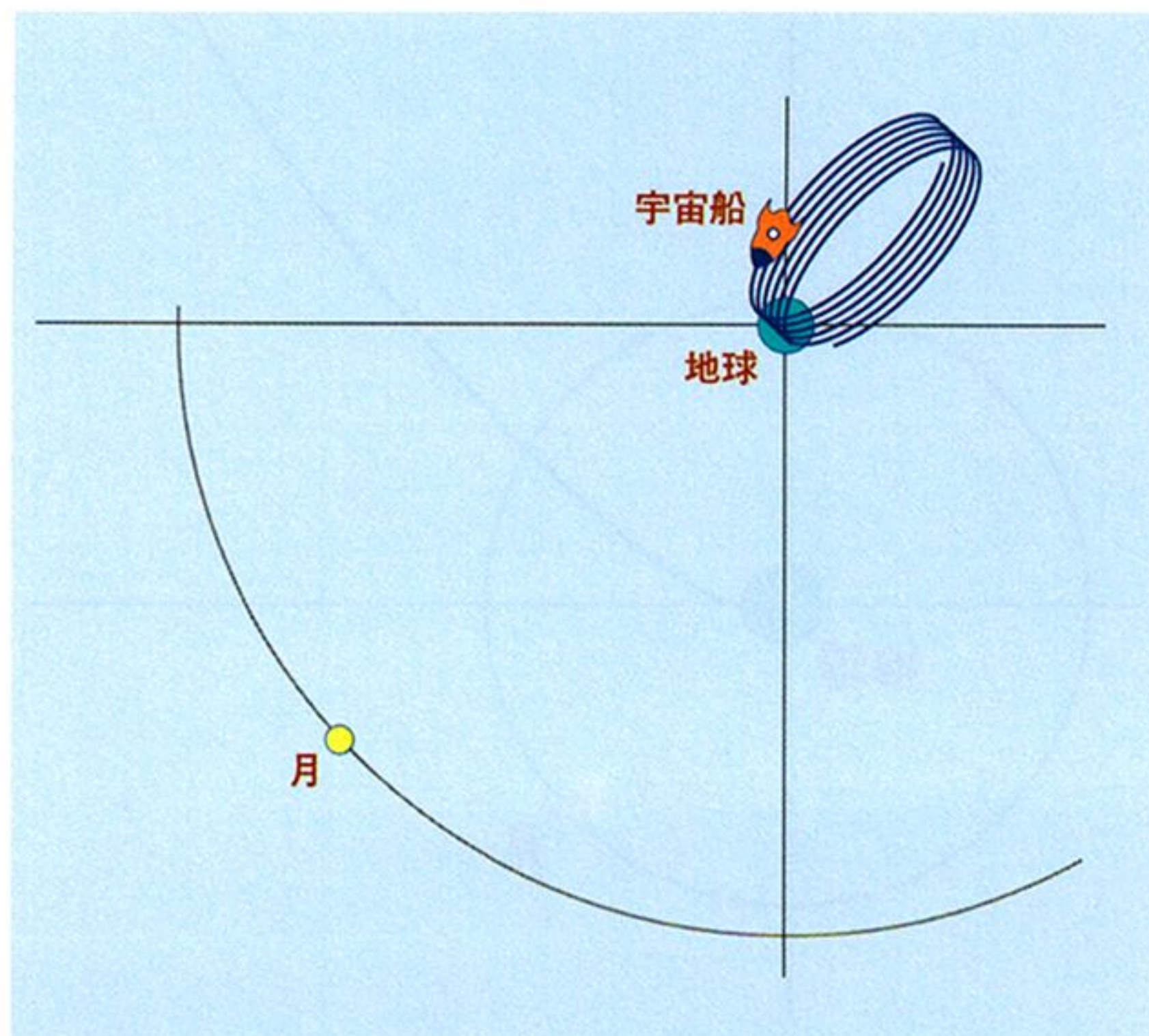



図3 ケース1。打ち上げ速度が足りないと地球の周りを回り続ける

(ケース2)地球から月に向かうが，
そのまま帰還できない場合

地球から月に向けて打ち上げたときに，なるべく月の近くの軌道を取ろうとすると，月を巡ったあと，地球からも月からも遠ざかっていく軌道になってしまう。これは，月の重力圏に入ったあと，月に引きつけられすぎて，加速度がついてしまい，その勢いでより遠くに飛ばされてしまうことによる。これを「スイングバイ」という。惑星探査機をより遠くに飛ばすために，行程の途中にある惑星の引力を使って加速させるときに用いられている手法だ。

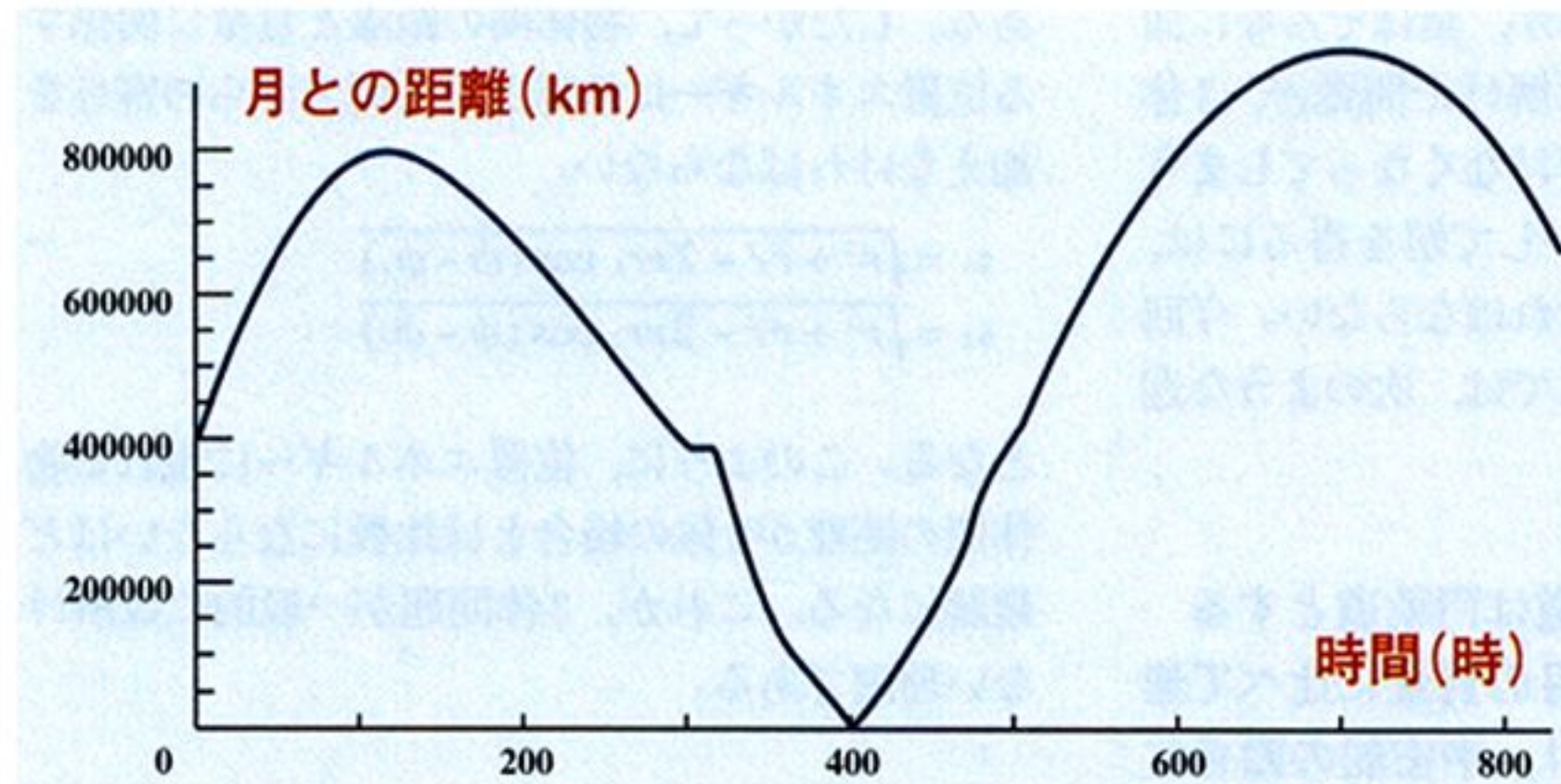
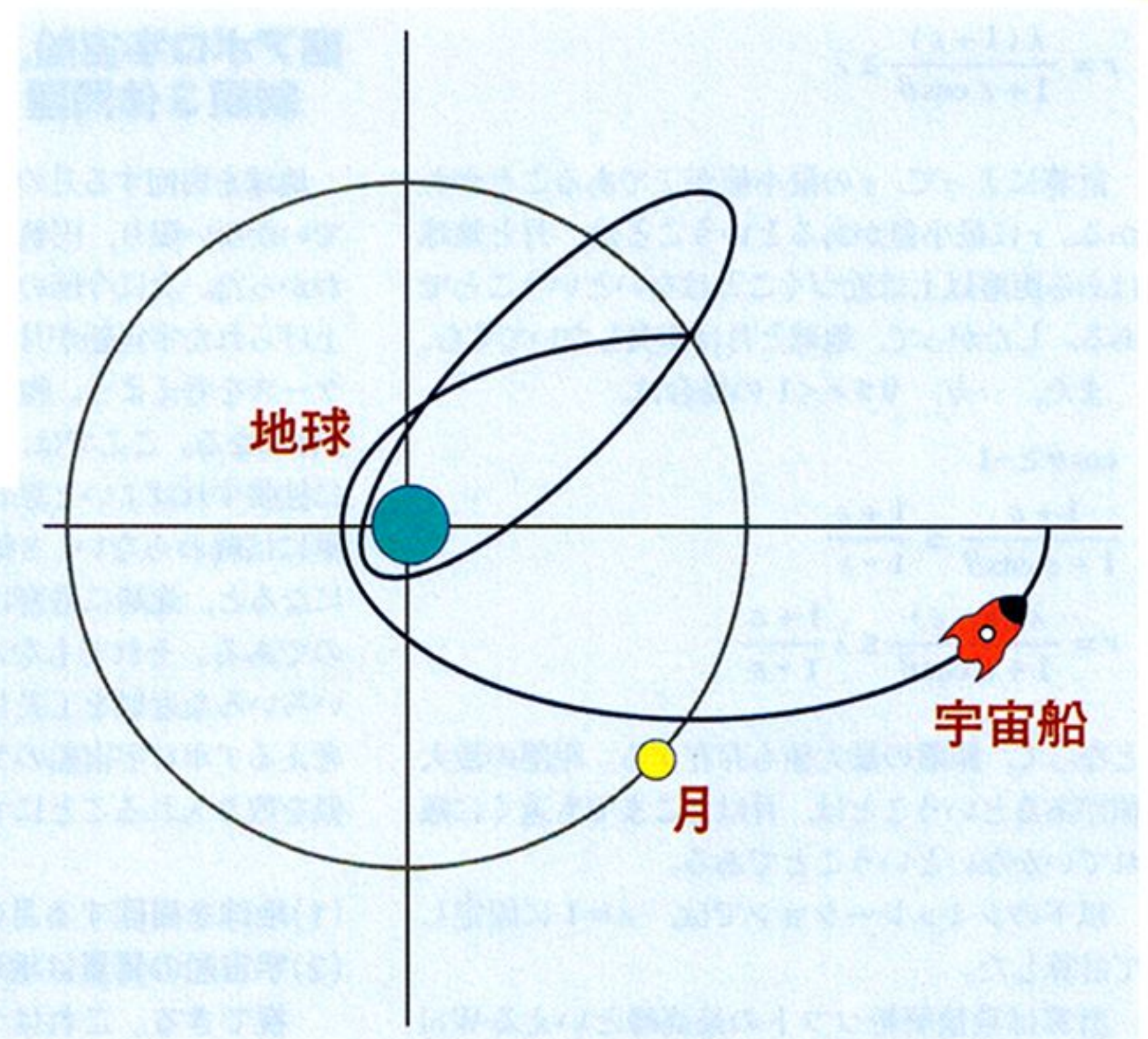



図4　ケース2。尽きに近づきすぎるとスイングバイで宇宙船が放り出される

(ケース3)地球から月に着陸できる場合

ケース2のように，地球を発射したときに初速度を決めたら，月の近くに軌道を取ろうとすればするほど，周回軌道から逸脱してしまうことになる。したがって，月に着陸するためには，月の引力圏に入ったら，減速してやらなければならないことになる。

(ケース4)月にまったく近づけない場合

初速度を大きく取りすぎると，月の重力圏内に入れずに，宇宙船はまったく遠くへ飛んでいってしまう。

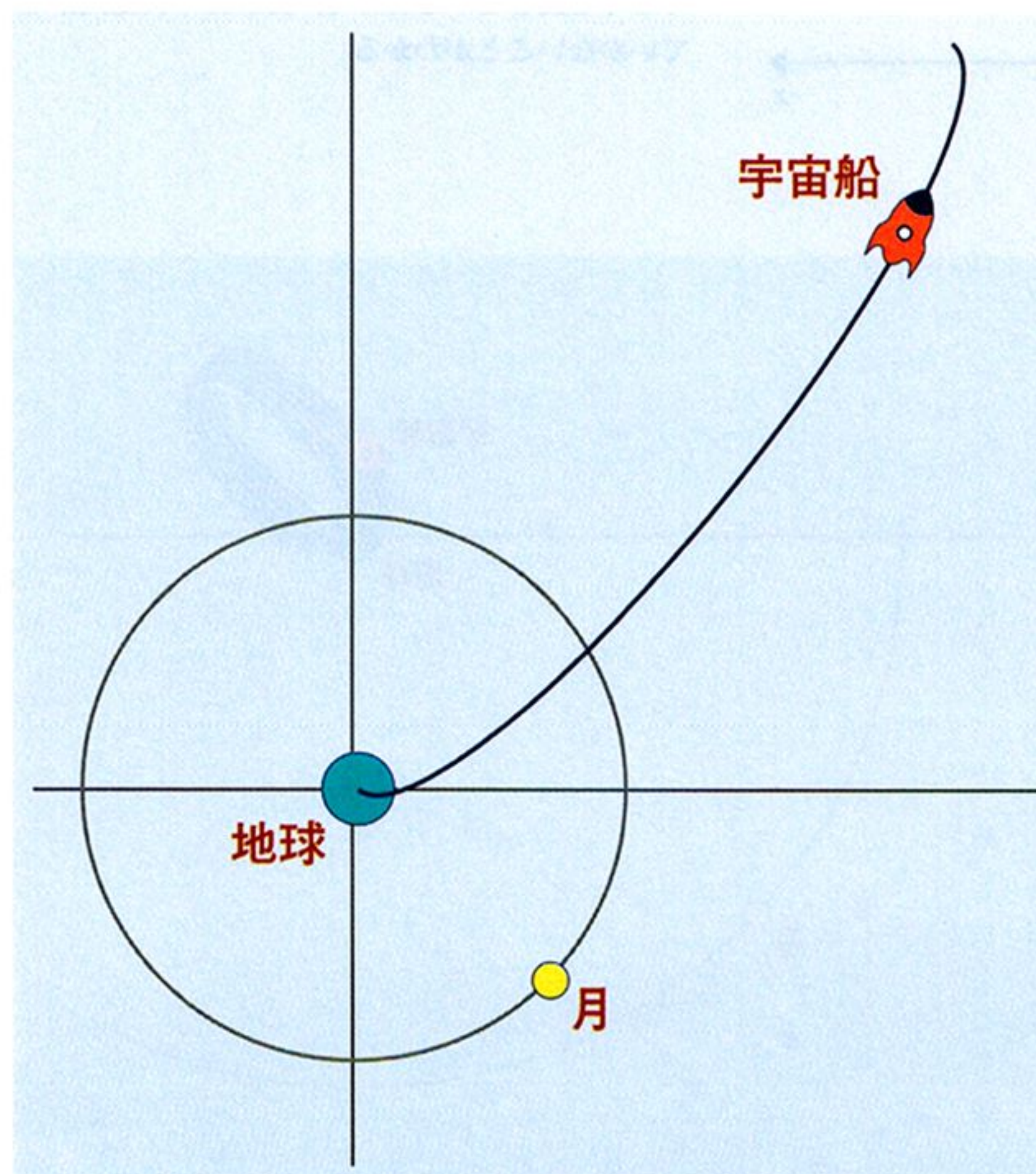



図5　ケース4。初速が大きすぎると月の重力を無視して外宇宙に向かってしまう

(ケース5)地球から月を周回して，
そのまま地球に帰還できる場合

アポロ宇宙船が月着陸に初めて成功したのは，11号であったが，それ以前の実験段階では月を周回してそのまま地球に帰還していた。地球から打ち上げる初速度をうまく設定すると，地球の引力圏から脱出し，月の周回軌道に移り変わって，また地球の周回軌道に戻ってくる。

映画にもなったアポロ13号では，ケース3のように月の周回軌道から月着陸を計画していたが，酸素タンクの爆発事故により，一刻も早く地球に戻ってこなければならなくなった。そこで，なんとかしてこのケース5の軌道に乗せることで月を巡って地球に生還することができたのである。

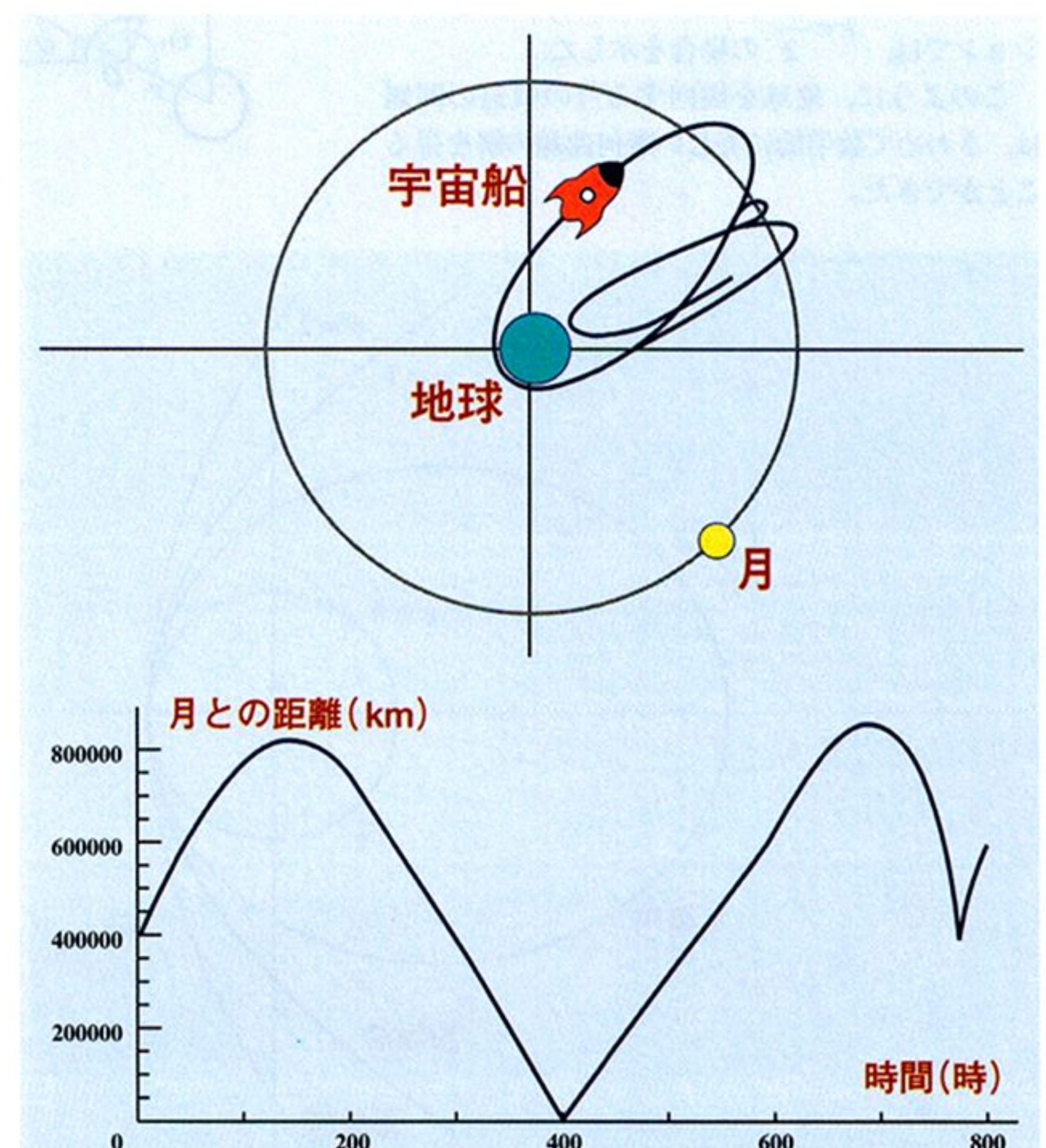



図6　ケース5。月を周回して地球に戻るための軌道

■ Parameter clear

```
In[1]:=  Clear[μ];
         Clear[m1];
         Clear[m2];
         Clear[g];
         Clear[ω];
         Clear[r1];
         Clear[r2];
         Clear[r0];
         Clear[k];
         Clear[m];
         Clear[r];
         Clear[x];
         Clear[y];
```

■ Coordinate transform

```
In[2]:=  x[t] = r[t] * Cos[φ[t]]
         y[t] = r[t] * Sin[φ[t]]

Out[2]=  Cos[φ[t]] r[t]

Out[3]=  r[t] Sin[φ[t]]

In[4]:=  φ1 = ω t;
         φ2 = ω t + π;

In[5]:=  x1[t] = r1 * Cos[φ1];
         y1[t] = r1 * Sin[φ1];

In[6]:=  x2[t] = r2 * Cos[φ2];
         y2[t] = r2 * Sin[φ2];

In[7]:=  s1 = Sqrt[r[t]^2 + r1^2 - 2 * r[t] * r1 * Cos[φ[t] - φ1]]

Out[7]=  Sqrt[r1^2 - 2 r1 Cos[t ω - φ[t]] r[t] + r[t]^2]

In[8]:=  s2 = Sqrt[r[t]^2 + r2^2 - 2 * r[t] * r2 * Cos[φ[t] - φ2]]

Out[8]=  Sqrt[r2^2 + 2 r2 Cos[t ω - φ[t]] r[t] + r[t]^2]

In[9]:=  xr[t] = r[t] * Cos[φ[t] - ω t]
         yr[t] = r[t] * Sin[φ[t] - ω t]

Out[9]=  Cos[t ω - φ[t]] r[t]

Out[10]= -r[t] Sin[t ω - φ[t]]
```

■ Hermite function

```
In[11]:= H = pr[t]^2/(2 μ) + pφ[t]^2/(2 μ r[t]^2) - g m1 μ/s1 - g m2 μ/s2

Out[11]= pr[t]^2/(2 μ) + pφ[t]^2/(2 μ r[t]^2)
         - g m1 μ/Sqrt[r1^2 - 2 r1 Cos[t ω - φ[t]] r[t] + r[t]^2]
         - g m2 μ/Sqrt[r2^2 + 2 r2 Cos[t ω - φ[t]] r[t] + r[t]^2]

In[12]:= dr = D[H, pr[t]]

Out[12]= pr[t]/μ

In[13]:= dφ = D[H, pφ[t]]

         General::spell1 : Possible spelling error: new symbol name "dφ" is similar to existing symbol "pφ".

Out[13]= pφ[t]/(μ r[t]^2)

In[14]:= dpr = -D[H, r[t]]

Out[14]= pφ[t]^2/(μ r[t]^3)
         - g m1 μ (-2 r1 Cos[t ω - φ[t]] + 2 r[t])/(2 (r1^2 - 2 r1 Cos[t ω - φ[t]] r[t] + r[t]^2)^(3/2))
         - g m2 μ (2 r2 Cos[t ω - φ[t]] + 2 r[t])/(2 (r2^2 + 2 r2 Cos[t ω - φ[t]] r[t] + r[t]^2)^(3/2))

In[15]:= dpφ = -D[H, φ[t]]

         General::spell : Possible spelling error: new symbol name "dpφ" is similar to existing symbols {dφ, pφ}.

Out[15]= g m1 r1 μ r[t] Sin[t ω - φ[t]]/(r1^2 - 2 r1 Cos[t ω - φ[t]] r[t] + r[t]^2)^(3/2)
         - g m2 r2 μ r[t] Sin[t ω - φ[t]]/(r2^2 + 2 r2 Cos[t ω - φ[t]] r[t] + r[t]^2)^(3/2)
```

■ Parameter set

■ Initial condition

```
In[20]:= rd = 6800;
         φd = 235 Degree;

         General::spell1 : Possible spelling error: new symbol name "φd" is similar to existing symbol "φ".

In[21]:= vd = 38900 ;
         ϕd = 325 Degree;

         General::spell1 : Possible spelling error: new symbol name "ϕd" is similar to existing symbol "φd".

In[22]:= td = 0;
```

■ Simulation

■ Solution

```
In[23]:= xd = rd Cos[φd];
         yd = rd Sin[φd];
         x0 = xd + r1 * Cos[ω td];
         y0 = yd + r1 * Sin[ω td];
         r0 = Sqrt[x0^2 + y0^2];
         φ0 = ArcTan[y0 / x0];
         vxd = vd Cos[ϕd];
         vyd = vd Sin[ϕd];
         vx0 = vxd - ω r1 * Sin[ω td];
         vy0 = vyd + ω r1 * Cos[ω td];
         pr0 = 1/r0 (x0 vx0 + y0 vy0);
         pφ0 = x0 vy0 - y0 vx0;

In[24]:= Clear[sol]

In[25]:= sol = NDSolve[{r'[t] == dr,
           φ'[t] == dφ,
           pr'[t] == dpr,
           pφ'[t] == dpφ,
           r[0] == r0, φ[0] == φ0, pr[0] == pr0,  pφ[0] == pφ0},
         {r, φ, pr, pφ}, {t, 0, 2000}, MaxSteps -> 10000,
          AccuracyGoal -> 10,
          PrecisionGoal -> 10,
          WorkingPrecision -> 16]

Out[25]= {{r -> InterpolatingFunction[{{0., 2000.}}, <>], φ -> InterpolatingFunction[{{0., 2000.}}, <>],
                                                pr -> InterpolatingFunction[{{0., 2000.}}, <>],
           pφ -> InterpolatingFunction[{{0., 2000.}}, <>]}}
```

■ Plot

```
In[26]:= satellite = ParametricPlot[Evaluate[{x[t], y[t] } /. sol], {t, 0, 800},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}];
```

```
In[27]:= satelliter = ParametricPlot[Evaluate[{xr[t], yr[t] } /. sol], {t, 0, 450},
         PlotRange -> All];

         General::spell1 : Possible spelling error: new symbol name "satelliter" is similar to existing symbol "satellite".
```

```
In[28]:= x1r[t] = r1 * Cos[φ1 - ω t]
         y1r[t] = r1 * Sin[φ1 - ω t]

Out[28]= 7529.41

Out[29]= 0

In[30]:= x2r[t] = r2 * Cos[φ2 - ω t]
         y2r[t] = r2 * Sin[φ2 - ω t]

Out[30]= -376471.

Out[31]= 0

In[32]:= x1r[t] - x2r[t]

Out[32]= 384000.

In[33]:= earth = ParametricPlot[Evaluate[{x1[t], y1[t] } /. sol], {t, 0, 800},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];

In[34]:= moon = ParametricPlot[Evaluate[{x2[t], y2[t] } /. sol], {t, 0, 800},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}, DisplayFunction -> Identity];

In[35]:= Show[satellite, earth, moon, Ticks -> None, AspectRatio -> 0.6];
```

```
In[36]:= Plot[Evaluate[{Sqrt[(x[t] - x2[t])^2 + (y[t] - y2[t])^2] } /. sol], {t, 0, 800},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}]
```

```
Out[36]= -Graphics-

In[37]:= Plot[Evaluate[{Sqrt[(x[t] - x1[t])^2 + (y[t] - y1[t])^2] } /. sol], {t, 0, 800},
         PlotRange -> All, PlotStyle -> {Thickness[0.01]}]
```

```
Out[37]= -Graphics-
```

図7　ケース2をMathmaticaで処理するとこんな感じになる

ネットワークゲームの地平へ

Diabloに見る ネットワークゲーム構成法

瀧 康史 *Taki Yasushi*

現状のネットワークでは帯域制限は厳しいものがあります。誰しも気になるのは，どのくらいのデータ量を通信できるのか，またはどのくらいのデータでどれだけのことができるのかということでしょう。ここではPCネットワークゲームの成功例であるDiabloを例に，ネットワークゲームの構成を考えてみましょう。

■久しぶりだから長い前書き

お久しぶりです。Oh!Xが休刊したとき，本当のところ再びOh!Xの記事を書くことになるとは思いませんでした。あれからずいぶんと月日がたち，私も長い学生生活に終止符を打ち，現在は普通の会社員（ゲーム関係）をやっています。

学生時代も半分以上ライターだったゆえ，当然忙しいかったわけですが，それは会社員になっても変わらず忙しい毎日を送っています。やれやれ。

当時は学生で，単位取り＆レポート＆原稿に追われ，きっと会社員になってライターをやめたら楽になるだろうと思ってたんですが，それは会社が楽だった場合だけですね。固定給なのに，仕事がヘビーなため即席に金にならない仕事が増えたから特にそう感じるのかもしれません。もともと性格がゲンキンでおこちゃまな分，今月の頑張りがすぐに返ってこないと，翌月踏んばれないという，そういう自分を最近発見している次第です。ああ，目先のニンジンがほしい。

とまぁ，人は変わって月日は流れてしまいました。

その間，世間と，自分が使うマシンはどんどん変わっていきました。Oh!Xが終わる頃の私は，どう頑張っても，やりたいことをX68000では実現できないジレンマを抱えていて，次第にX68000から心が離れていっていた時期でした。結論からいうと，Xが終わったら個人的にX68000を使う気も起きなかったので，すぐにX68000を売り払い，以後はいわゆる標準機であるPCを使い続けています。X68000をその後も使い続けている人は，おそらくX68000でもまだやりたいことが残っているのか，それとも単純にX68000が好きなのか，シャープが好きなのか，その人にとってやりたいことができるマシンなのか，そういうことなのでしょう。いまだってX68000で十分な遊びもありますから。それはそれが高級であるとか，低級であるとかではなくて，単に遊びのベクトルがX68000で包含できるかどうかだけなのでしょう（編注：ほかに触りたくなるマシンがないという場合もあるだろうなあ）。

しかし，自分がやりたいことと，そのハードでできることは違います。ハードにあわせてやりたいという欲求を縮めていきたくはないですし。そもそも，やりたいことができないマシンにいつまでも固執して，自分のやりたいことを狭めていくのは，Xらしくないってのが私のいまのXの解釈です。そうでなければXの復刊は意味がないと思いますし。

もちろん，X68000を捨てたいまでも，あのマシンを選んだ自分は間違ってなかったと，胸を張っていえるのは間違いない事実です。

当時のX68000は2Dゲームの時代の産物で，2Dのゲームを作るに当たっては，非常にGoodな箱庭でした。X68000を使うことで，2Dゲームは表示色とかスプライト数に制限はあったものの，ひととおりのものは作れました。実際，パソコン通信や同人ソフトには市販ソフトやアーケードのパクリがたくさんあり，模倣することでX68000がよい教材になっていきました。

そのうちハードが進歩して3Dができるようになり，3Dという遊びができてきました。残念ながらゲームをユーザーが作れる環境は一歩遅れ，最近やっとDirectXで揃い始めたというところです。実際は運よくOpenGLを先に触り始めることができていた人たちが，すでに3Dに対する知識を深めて，バリバリ現役でゲームを作っているわけです。

3Dになることで，ゲームに必要な数学が大学レベルになり，ますますゲームを作るために必要な知識が倍増していき，それ相応の根性が必要になってきました。いまでは，2Dには2Dの遊びがありますが，3Dには3Dの遊び方ができています。もちろん，3Dにする意味がまったくない3Dゲームがこの世にはあふれていますが，3Dじゃなければできない遊びもいっぱいあるのは，皆さんも周知のことだと思います。

先にもいったとおり，現在のゲームは大半が3Dを前提にして作られます。一部の幸運な人たちは，それ以前からOpenGLを使って3Dの理解を深めていましたが，彼らにとってOpenGLを使えるSGI環境というのがいい箱庭になっていたのは事実です。そういう環境にやっとPCが追いついてきました。最近のビデオカードなら，文句がないぐらいの性能が出てきます。そういう意味でいうなら，3Dをしたい人には，やっと箱庭が揃い始めました。最初にSGIとOpenGLを触り始めた人たちには大きく水を開けられていますが，結局のところ，個人個人の頑張り次第ってところでしょう。

これからまた，3Dゲームは進歩していくと思われます。しかし，3Dゲームの次にくるのはなにがあるのでしょうか？

うまい日本語が思いつかないのですが，厚みのあるゲームシステムがひとつ目かなと思います。語弊があるので平たく説明しますが，3Dゲームの大半は，アーケードや，コンシューマゲーム機が先行していた理由もあって，少ない記憶容量の下で作られています。CD-ROMにしてもROMにしても，ROMは決められたものしか保存できません。ですから，決められたヒーローをロールプレイングする，ヒロイックファンタジーものとか，決められた遊びしかできず，映像と効果だけ綺麗な，比喩をすればハリウッド映画のようなゲームだけが進歩してきたともいえます。

PCゲームはもともと，地味だけど深みがあるものが多かったと思います。理由はRAMがたくさんあり，CPUパワーに比較的余裕があるということにあるでしょう。厚みというかなんというか，決められたヒーローではなくて自分をロールプレイしたり，不確定な要因や深みをつけることができます。それに加えて3Dを強化したことにより，地味だった映像面が解決してきました。

記憶容量の増加とCPUパワーの増大による恩恵の一方で，次の世代で昇華していくであろうゲームのエッセンスにネットワークがあります。ネットワークはどのようにゲームを面白くしてくれるのでしょうか。

■ゲームのネットワーク化について

コンピュータのゲームは一人遊戯のものが大半を占めるのですが，もともとゲームというものは複数人数で行うものでした。将棋，囲碁から，さまざまなスポーツまで，古典的なゲームは，場の中で複数人数でいわば対戦してゲームを楽しみます。

コンピュータゲームの大きな特徴は，人間相手ではなく，コンピュータが相手になることです。

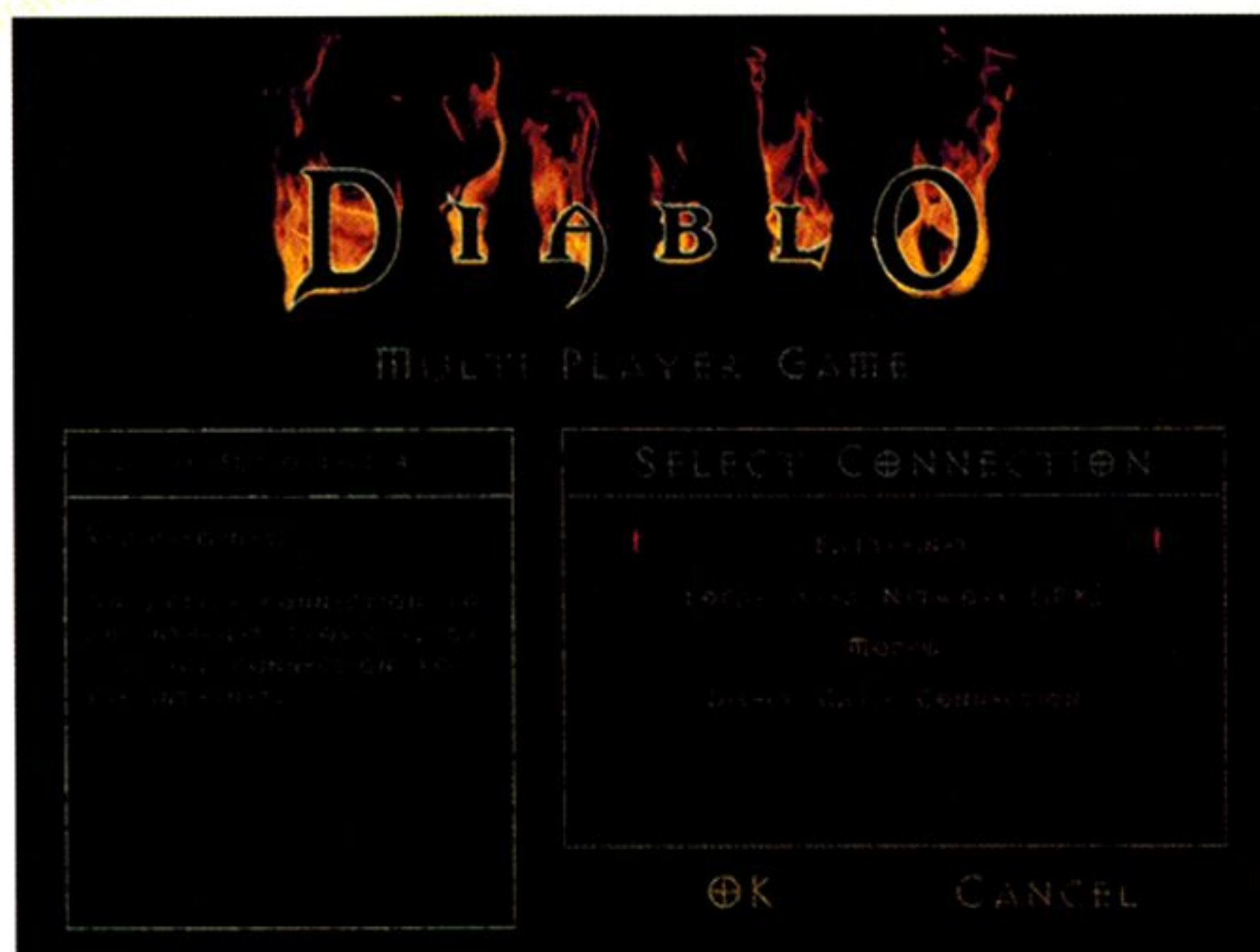



図1 ピアトゥピアネットワークでの典型的成功例といえるDiablo

時代は変わって文化は湾曲し，代わりにコンピュータと対戦するものではなく，電子的アトラクションを遊ぶものへとコンピュータゲームはジャンルを広げていきました。

コンピュータと闘うということは即ち，コンピュータのアルゴリズム（プログラム）と闘うことを意味します。AIが未完成であることからわかるように，現時点では残念ながら，所詮コンピュータはコンピュータであり，人間ではありません。反論もあるでしょうが，現実的に考えて現時点においては，アルゴリズムと闘うよりも，人間と闘ったほうが面白いと思うのは当然でしょう。まして隣人と闘うときにある駆け引きは，「所詮人間は戦いの中に身を置いてしまう生き物なのか？」と，妙な錯覚を起こすほど面白かったりします。

コンピュータゲームにおいても，対戦はゲームを熱くしてくれます。しかし，同じゲーム機にアクセスできる間柄でしか，ゲームは一緒にプレイできません。それゆえ，対戦ゲームはお互いに同じものが家にあっても，出先のゲームセンターでプレイしてしまいます。友人と遊ぶゲームがメインであるゲームは，逆に友人とアクセスできないと，面白さがほとんど失われてしまいます。

現在はコンピュータの対戦格闘ゲームはたくさんあるので，友達とハマってしまったゲームがたまたま違うと，楽しむことができなくなってしまいます。むしろ同じゲームにハマっている人を探すことが重要になるわけです。

ネットワークはコンピュータ同士を接続しますが，このときゲームの世界も接続されます。ゲームがそこそこにヒットしていて，プレイヤーの数が見込めるなら，隣人友人はプレイしてくれなくても，ネットワークの先の誰かはプレイしているかもしれません。ネット友達がプレイしてくれるなら，その対戦は成り立ちます。

また，圧倒的多数に，ネットワークゲーム仲間は広がるので，対戦のバリエーションもどんどん増えていきます。対戦ゲームをネットワークベースに乗せるというのが，まずゲームのネットワーク化の方法論のひとつでしょう。

ネットワークの利用方法はひとつだけではありません。とにかく，“繋がる”レイヤーだけを提供するのがネットワークですから，その上でいろんなことができます。たとえば，Ultima Onlineは，サーバを用意し，その内部でゲームが動きます。厳密には違う部分があるのでしょうが，基本的なスタンスはサーバ上でゲームが動き，プレイヤーのマシンではサーバにアクセスするためのクライアントプログラムが動いています。クライアントプログラムは，プレイヤー寄りの話をするなら，いわばUOの世界の窓のようなものです。もっと，作り手の側に立った話をするならば，クライアントプログラムはフロントエンドプロセッサにすぎません。ユーザーインタフェイスなどの処理を含めてのフロントエンドプロセスを，クライアント上で行い，ゲームはサーバで運営されていきます。

ゲーム上ではこれにより，ひとつの世界の中で物事が進行していくような，仮想空間ができます。これらはネットワークでないとできない遊びのひとつです。

さて，ゲームのネットワーク化の意義を考えていくのは，これ以上は意味がないのでやめて，先に進みましょう。

■ゲームにおけるネットワークの組み方

ゲームプログラムの繋がり次第で，ゲームデザインも大きく変わってきます。そして，接続の方法によってできることとできないことが，明確になってきます。

2つのゲームプログラムをネットワーク接続するということは，ゲームにおける世界を接続するということを意味します。これは，ある種のドグマのようなもので，のちにその否定を行いますが，基本的にはこういう考え方です。

そして，互いが互いの世界を同時化することで，2つのゲームにおける世界に矛盾が生じなくなり，対戦が成り立ちます。

逆に，昔の対戦ゲームにあった，1台のゲームプログラムに2本のジョイスティックが繋がっているという状態では，世界はひとつなので同時性を意識する必要性はありません。しかし，2つのマシンを接続してゲームを成り立たせるためには，大前提として2つのゲームプログラムで同じ世界を作らないことにはゲームが成り立ちません。たとえば将棋なら，プレイヤー1のゲーム世界とプレイヤー2のゲーム世界の中で，“金”のいる場所が違うという事態が起き，リアルタイムゲームでプレイヤーAが打った弾が，プレイヤーBの世界には存在しないといったことがあるとゲームは途端に破綻してしまいます。

ネットワークゲームを作るということは，2つの離れたゲーム世界において同時性を保たなくてはならないのです。

さて，ネットワークの組み方にはいろいろありますが，現在，ネットワークゲームで使われているものは，大まかに分けると下記のとおりです。

i. ピアトゥピア方式
ii. サーバ集中型
iii. ピアトゥピア＋補助サーバ型

それぞれの方式ごとに特徴と問題性があります。それゆえに，それぞれの方式でできるゲーム，できないゲームが具体的になってきます。

ピアトゥピア形式は，互いに互いのデータを送りあう方式です。基本的に，1対1でデータを送りあう方式であるため，2人でしか通信しあえません。3人で行うためには，相互に1対1の転送をするので3接続，4人だと6接続が必要になります。Diabloがこの方式をとっています。

サーバ集中型は，サーバのパフォーマンスによってできることが異なりますが，基本的にはいったんクライアントのデータをサーバに集め，再配信することでゲームを進行させます。Ultima Onlineがこの形式です。パケットをただ集めて揃えて返すだけのサーバをリフレクタと呼んだりします。

ピアトゥピアに補助サーバを加えた形もあります。結局ピアトゥピアなのですが，ただのピアトゥピアではメンバーを集めることができないので，別サーバを立てて，そこでマッチメイキングをしてからゲームが始まります。

ゲームが始まったら，サーバから切り離されてピアトゥピアになります。Diabloのbattle.netがこの方式です。

おおまかにこれぐらいなのですが，方法論とリソースの大きさでゲームの側面はぐっと変わってきます。

■実例を見る

ネットワークゲームの草分けであるDiabloのパケットをパケットモニタで見ながら，いろいろ推察してみることにしました。現実に流れているパケットは，図3のとおり，なにを送っているのかほとんど意味不明で，なにをすればデータ通信が増えるのかも明確にはわかりませんでした。

とりあえずは，実際の通信データ量を見るために，Excelでグラフにしてみました（図4）。

パケットを見る限り，秒間40のパケットを送りあっています。おそらく，ゲームが20フレーム/秒で動いていて，1Syncあたり1パケットを送っているのでしょう。Diabloは3人プレイをする場合，ピアツーピアを6接続し，4人の場合は12接続します。したがって，途中でひとり入ると，HUBを通るパケットは3人で秒間120パケットになります。

秒間40パケットにきっちり安定していない理由はわかりません。マシンの速度が互いに違いすぎること（PentiumⅡ/300のデスクトップとPentium/133のノートPC）があるので，そのせいかもしれませんし，あるいは1パケットあたりの制限（1560バイト）に当たって，ゲーム中の内部イベントが増えたときに，一時的に複数のパケットを送りあっているのかもしれません。

なにをすればデータ量が増えるか見るために，いくつか実験してみました。表示オブジェクト数はどうでしょうか？　とりあえず，町でものを置いたり，取ったりしても，特別データが増えているようには見えません。チャットでデータをたくさん送りあっても，ほとんど違いはわかりませんし，チャットのテキストもそのまま送られている様子はありません。

試しにこれ以上地面に置けなくなるぐらい，たくさんのものを置いてみます。Diabloでは，インベントリーのお金の上で右クリックをすると，いくら置くかダイアログが出てくるので，1ずつ置いていきました。1レベル（フロア？）あたりに最大に置ける数は決まっているので，もう置けないほど置いてみましたが，わずかになんとなくパケットが増えただけのような感じがします。その程度です。

基本的に1Syncあたり，1パケットを送りあうので，ひとりがレベル間の移動をすると，パケット数が20減ります。途中，Diabloによくある，一瞬止まる現象が起きると，パケット数も当然減っています。面白いのは，そのあとで跳ね上がることです。Diabloをプレイしたことがある人はわかると思いますが，再度復活したときに，一瞬高速に動くのが，この状態です。

レベル1ダンジョンで敵に囲まれると，よくあるように徐々にデータ量は増えていきます。協力プレイでお互いに周りの敵と戦いあうと，さらにデータ量は増えます。

データ量とパケットを見てみると，だいたいデータ量は4KB/secぐらいを前後していることがわかります。ゲーム中の1Sync，つまり1パケットあたり4KB÷40＝100バイト程度のデータしか送信してないことがわかります。

このサンプリングテストではLANの中で10BASEのIPXを用いた転送で行いました。2プレイで最大4KB/secを超えるということは，4プレイでは最大2KB×12＝24KB/secの転送を行わなくてはなりません。これは，28.8kモデムの理論値でぎりぎり転送できる程度です。インターネットでは実効値はころころ変わるため，実際には途切れたりすることもあるでしょう。

実際のDiabloでどのようなデータを転送しているのかは，わかりません。しかし，1Syncで転送できるデータが100バイトのところを見ると，データはかなり少なめであることがわかります。

ネットワークのコリジョンを避けるために，ひとつのマシンをサーバに見立てて，それをリフレクタとして使い，いったん入力をそのサーバに集めてから再配信する方法も考えられます。しかしその場合，サーバマシンが落ちると全滅してしまいます。Diabloの場合，Battle.netにはマッチメイキングとしてのチャットサーバがありますが，実際にはゲーム中のサーバはありません。当然，LANでのIPX通信においても，サーバはありません。まして，こっそり誰かのマシンをサーバに見立てる機能を入れたとしても，そのマシンが突然ハングアップすると，すべてのゲームは落ちてしまい，非常にゲームが不安定になってしまいます（代替してほかのサーバが立つように作れればいいのですが）。

ネットワークでゲームの同期を図る際，複数のマシンのそれぞれのユーザーの入力をいったん同期する方法論があります。もちろん，このシステムを使う際には，すべてのコンピュータにおいて同じ入力があったとき，同じ動きをするようにできていることが前提です。

やり方はいくつかあるでしょうが，ひとつの方法としては，表1に書いてあるような手順を踏まなくてはなりません。これは，完全にすべてのマシンのプログラム中のゲーム世界を同期でき，しかもデータ量が少ない一方で，キー入力をいったん転送しあってまとめ上げ，互いに処理をしあう

図3　Diabloのネットワーク対戦時に流れているパケット例

```
774 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.266 0.063.000  2007/07/17 10:57:10 午後 D=0x17E0 S=0x17E0
775 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.296 0.030.990  2007/07/17 10:57:10 午後 NetWare code=F0CD(hex)
776 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.359 0.062.010  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
777 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.390 0.031.995  2007/07/17 10:57:11 午後 NetWare code=F8C3(hex)
778 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.452 0.061.995  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
779 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.452 0.000.000  2007/07/17 10:57:11 午後 NetWare code=BD90(hex)
780 0.008098303F3F  0.00C04F8EE4A4  NCP  67 000:00:34.468 0.016.005  2007/07/17 10:57:11 午後 NetWare code=BA8F(hex)
781 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.483 0.015.000  2007/07/17 10:57:11 午後 NetWare code=01B9(hex)
782 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.563 0.079.005  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
783 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.578 0.015.000  2007/07/17 10:57:11 午後 NetWare code=09AF(hex)
784 0.00C04F8EE4A4  0.008098303F3F  NCP  68 000:00:34.608 0.030.990  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
785 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.656 0.047.010  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
786 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.687 0.031.995  2007/07/17 10:57:11 午後 NetWare code=11A5(hex)
787 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.765 0.078.000  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
788 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.797 0.031.005  2007/07/17 10:57:11 午後 NetWare code=199B(hex)
789 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.858 0.061.995  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
790 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.858 0.000.000  2007/07/17 10:57:11 午後 NetWare code=C586(hex)
791 0.008098303F3F  0.00C04F8EE4A4  NCP  67 000:00:34.875 0.016.005  2007/07/17 10:57:11 午後 NetWare code=C285(hex)
792 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.890 0.015.990  2007/07/17 10:57:11 午後 NetWare code=2191(hex)
793 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:34.953 0.062.010  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
794 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:34.968 0.015.990  2007/07/17 10:57:11 午後 NetWare code=2987(hex)
795 0.00C04F8EE4A4  0.008098303F3F  NCP  68 000:00:35.016 0.047.010  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
796 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.062 0.046.995  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
797 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.077 0.015.000  2007/07/17 10:57:11 午後 NetWare code=317D(hex)
798 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.155 0.078.000  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
799 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.187 0.031.995  2007/07/17 10:57:11 午後 NetWare code=3973(hex)
800 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.265 0.078.000  2007/07/17 10:57:11 午後 D=0x17E0 S=0x17E0
801 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.265 0.000.000  2007/07/17 10:57:11 午後 NetWare code=CD7C(hex)
802 0.008098303F3F  0.00C04F8EE4A4  NCP  67 000:00:35.265 0.000.000  2007/07/17 10:57:11 午後 NetWare code=CA7B(hex)
803 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.296 0.031.005  2007/07/17 10:57:11 午後 NetWare code=4169(hex)
804 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.358 0.061.995  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
805 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.391 0.032.010  2007/07/17 10:57:12 午後 NetWare code=495F(hex)
806 0.00C04F8EE4A4  0.008098303F3F  NCP  68 000:00:35.406 0.015.000  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
807 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.452 0.046.995  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
808 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.469 0.016.005  2007/07/17 10:57:12 午後 NetWare code=5155(hex)
809 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.562 0.093.990  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
810 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.577 0.015.000  2007/07/17 10:57:12 午後 NetWare code=594B(hex)
811 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.655 0.078.000  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
812 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.655 0.000.000  2007/07/17 10:57:12 午後 NetWare code=D572(hex)
813 0.008098303F3F  0.00C04F8EE4A4  NCP  67 000:00:35.671 0.016.005  2007/07/17 10:57:12 午後 NetWare code=D271(hex)
814 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.671 0.000.000  2007/07/17 10:57:12 午後 NetWare code=6141(hex)
815 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.766 0.094.005  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
816 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.781 0.015.000  2007/07/17 10:57:12 午後 NetWare code=6937(hex)
817 0.00C04F8EE4A4  0.008098303F3F  NCP  68 000:00:35.812 0.031.995  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
818 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.859 0.046.005  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
819 0.008098303F3F  0.00C04F8EE4A4  NCP  64 000:00:35.890 0.031.995  2007/07/17 10:57:12 午後 NetWare code=712D(hex)
820 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:35.952 0.061.995  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
821 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:36.063 0.110.010  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
822 0.00C04F8EE4A4  0.008098303F3F  NCP  82 000:00:36.156 0.093.000  2007/07/17 10:57:12 午後 D=0x17E0 S=0x17E0
```

図2　ネットワーク経由でゲーム中……

Diablo Network通信データ量

byte/sec
packet/sec

Duel中

Data

10000
1000
100
10
1

2プレイヤーでダンジョン1階で敵に囲まれて死にそうになったところ

一瞬パケット数が下がるところは、片方のプレイヤーが、レベル(ダンジョンの階)を移動しているところ

片方のマシンが一瞬重くなった(スワップ?) 道連れでもうひとつのマシンも一瞬止まる

	sec	128	252	377	501	626	750	875	999	1124	1248	1373	1497
byte/sec	0	186	432	410	462	324	302	244	410	429	397	295	293
packet/sec	0	26	41	41	47	38	41	29	41	46	44	36	40

パケット数は40を前後する20フレームで，相互に送りあうので40なのだろう

片方が死んだ。これ以後，死んだ方は赤画面でしばらく放置

図4　ゲーム中の通信パケット数とデータ量

ため，ネットワークの遅延の分，プレイヤーのマシンでさえも，入力に対する動作の遅延が起きてしまいます。これは，リアルタイムゲームではネックにもなりえます。

残念なことにIPXを用いたLANの転送であったとしても，実際にはネットワークのコリジョンが発生し，20packet/secでさえ転送しあえないこともあります。これは，グラフを見ればすぐにわかるでしょう。現実には，マシンのひとつのHDDのスワップやその他のネットワークの都合で，意外にもきっちり送りあえないことがあることがわかります。

もちろん，この方法論ではどれかひとつのコンピュータのCPUの速度が遅すぎて遅延を起こすと，すべてのマシンでフレーム落ちや，停止が起きてしまいます。ゲームは，もっとも遅い回線を持ったもっとも遅いマシンにあわせてしか進行しません。

現実的に考えて，インターネット経由では，この方法論を用いてリアルタイムにシステムを機能させることは難しいでしょう。特にPCでは，プレイヤーの用意するコンピュータのパフォーマンスが違いすぎて，現実的な方法論ではありません。

さらに，大規模なアプリケーションになると，キー入力だけでゲーム世界を同期させることは，理論的には可能であっても，これを実現するにはかなり技術力と注意力が必要になってしまいます。すべての情報をきっちり送りあう方法論もありますが，これにはデータ量がいたずらに多くなるという問題が発生してしまいます。

ネットワークの遅延があっても，ゲームの遅延があまり直感的に感じられないように，システムを構成するにはどうすればいいか？　ひと言でいえば，それは簡単なことで，すべての動作を1プレイでゲームするのと同じように1Sync中に行い，あとで送りあってお互いに状態を補正しあえばよいことになります。

しかし，これは考えてみればわかるとおり，フレーム単位で時間を逆行する動作であるので，同期がきちんとできるかは泥縄的に難しく，きちんと訂正しあえるかわかりません。なかでも大きな問題は，プレイヤー同士の衝突です。基本的には入力が同じならば同じような動作をするようにすることはもちろんなので，コンピュータが動かすキャラクターは，プレイヤーが干渉しない限り，どちらのマシンも一定の動きをするようにします。しかし，コンピュータ（というか，システム）から見て，未知なものはほかならぬプレイヤーの動きになります。ある程度の予測の範囲にはいますが，すべての予測することは，ほとんどのゲームではできません（もちろん，予測できるゲームシステムであるなら，それを最大限に生かしてゲームを作ればよいのですが）。

Diabloのシステムにしても，プレイヤーの動きの予測は同じく難しい作業です。当然，最大に難しい処理は，プレイヤー同士のDuel（対戦）でしょう。Duelをすると秒間データ転送量が増えましたが，その瞬間に2つのプレイヤーの画面を見ると，それぞれの場所は微妙にずれています。特に斬りあって動きあうと大いにずれてしまいます。ウォーリアーを使いあって，お互いに斬りあうと，プレイヤーAのコンピュータAの画面では，見かけ上，こちらの攻撃しか当たってないように見えるのに，プレイヤーAのキャラクターのHPがどんどん減っていきます。あれって思った

表1　入力の同期

	Computer A	Computer B	Computer C
1Sync	キー入力	キー入力	キー入力
	BとCに転送	AとCに転送	AとBに転送
2Sync	B，Cから受理	A，Cから受理	A，Bから受理
	それぞれのコンピュータでABCの入力を合わせて，プレイヤキャラクターABCをそれぞれのコンピュータ上で動かす		
	キー入力	キー入力	キー入力
	BとCに転送	AとCに転送	AとBに転送
3Sync	以後2Syncと同じことをする		

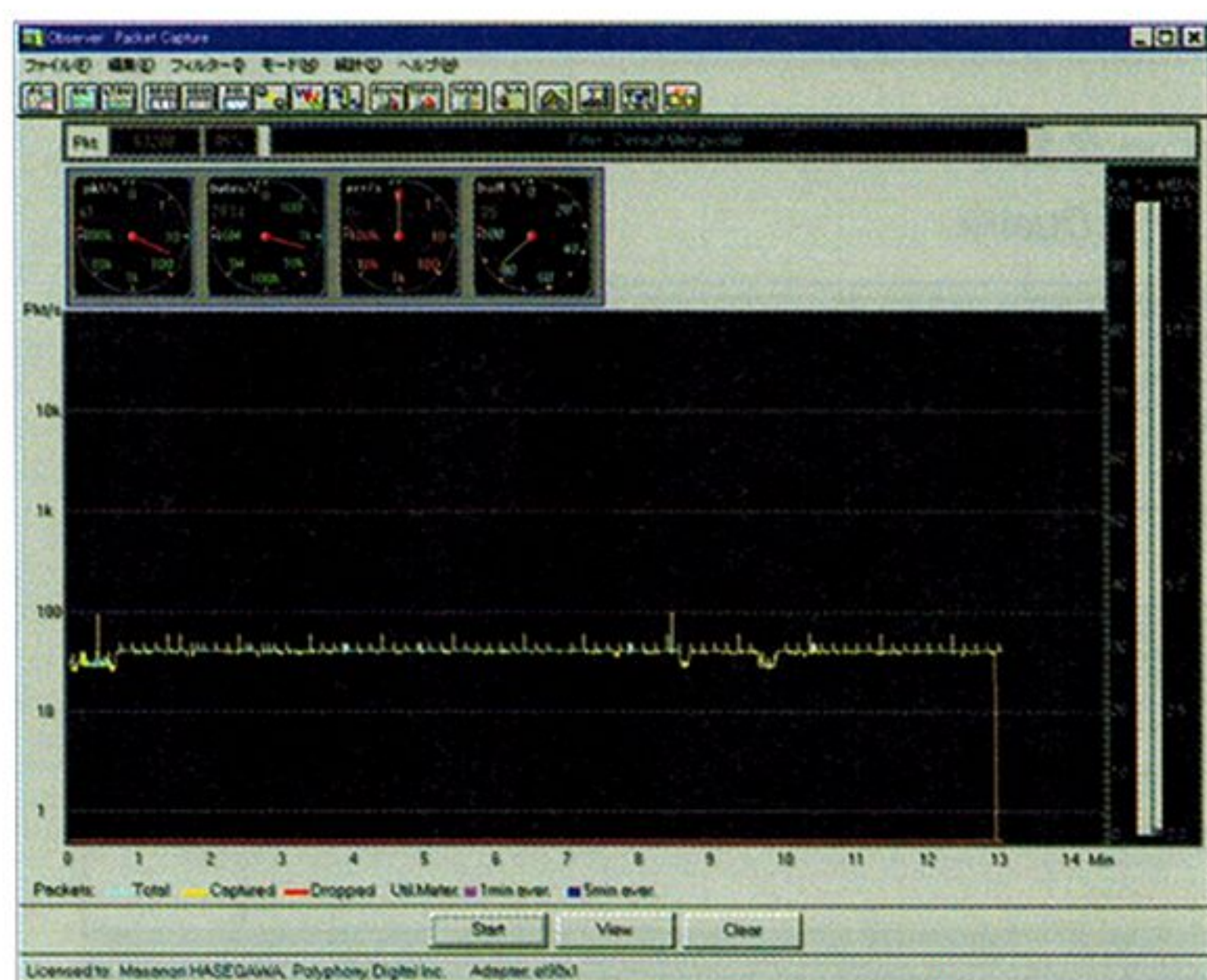



図5　パケットモニタ。このようなネットワークツールでデータ量の変動や安定性を確認できる

ことがありませんか？　その瞬間，プレイヤーBのコンピュータBの画面を見ても同じことが起きていて，プレイヤーBのキャラクターの攻撃だけが当たっているように見えるのに，HPはどんどん減っていきます。どちらも対戦相手のやられグラフィックしか見えていないのに……です。

おそらく，2つのコンピュータの内部では完全に同期が取れていないのでしょう。基本的にはそれぞれのコンピュータの上で当たり判定を行い，その結果のみを与えあいます。もちろん，複数の問題がほかにあるので一概にはいえませんが，乱暴にいえばプレイヤーAはプレイヤーBにいくつのダメージを与えた。プレイヤーBはプレイヤーAにいくつのダメージを与えたというのを送りあったところで，現実にダメージの算出をお互いにしているのではないかと予測できます。これなら，データは少なくて済みます。

キャラクター同士が協力しあい，強い「ユニーク敵」を倒していたという場合も，お互いのコンピュータ上でダメージを算出し，結果だけをお互いのマシンに送りあいます。そのユニーク敵のHPがある瞬間において，整合性が取れてなかったとしても，そのユニーク敵が〝死んだ瞬間〟，〝死んだ〟ことを送りあえば，ゲームは十分面白くできます。

厳密には，〝嘘〟といえるのかもしれませんが，プレイヤーの見た目のゲーム世界で，面白くできていれば，それはマルです。先に，ネットワークゲームにある種のドグマがあり，それは2つの世界を同期するといいました。これは可能であるならばそうすればよいのですが，時代性を含んだネットワーク環境において，最適なシステムをデザインする際，プレイヤーの見た目の世界の同期が取れていれば内部の同期はなんだっていいということです。

いま，さらっと流しましたが，インターネットという瞬間瞬間で信頼性が低いネットワークでもゲームが成り立ち，かつマシン同士の性能に大きく隔たりがあっても問題がなく，高負荷をかけられるサーバを維持するメンテナンス料がなくて済む。このような条件を満たせば，非常にバランスよくネットワークゲームがデザインされているといえます。

■ケーススタディ

Diabloを考察した結果，状況に応じて構築できるシステムが大きく変化することがわかりました。実際，たとえばユーザーメイド（個人やサークルなど）でネットワークゲームを作る際に，現実のインターネットでどの程度のことができるのか，少し考えてみましょう。

いろいろなケースが考えられますが，個人でできる範囲ということで，次の条件に限定してみました。

1.LANの中でサーバなし
2.LANの中でサーバあり
3.モデム直結でサーバなし
4.モデムでインターネットを通してサーバなし
5.モデムでインターネットを通してサーバあり
6.サーバのみOCNとか128kISDN

それぞれのバンド幅を，かなり大雑把に計算して，なにができるのか考えてみましょう。

1.LANの中でサーバなし

LANは10BASEだとします。100BASEだと単純に転送量10倍だと考えてください（Windowsなどでは実際にはそこまでいかない）。10BASEは10Mbpsの通信速度が出ます。20フレームのゲームを作ったとして，フレームあたり1パケット転送をするソフトだとします。秒間20パケットのデータ転送をすると，1パケットあたり，10M÷20=500Kビット/packet，つまり理論値において50KB/packetのデータ転送ができます。同一HUBを通してゲームをするなら，この値はそのHUBを通るすべてのデータの総和です。2人対戦なら，互いに送りあうことになるので，この2倍の転送量を要求します。したがって，25KB/packetのデータまで送れます。もっとも，送りあうということは，ネットワークのコリジョンが発生しまくるということなので，実際にはこの半分ぐらいの10KB/packetまでしか，実効値では使えません。まあ，これだけあれば，結構データを送りあうことはできるでしょう。ゲーム上の動的パラメータを送りあってもそこそこ成り立ってしまうかもしれません。2人なら入力を送りあって，入力にネットワーク分の遅延があっても許せるでしょう。

ピアトゥピアなら，人数が増えていくに従って，お互いに送りあわねばならないデータの量はどんどん増えていってしまいます。3人目からは，入力を同期してってわけにもいかなくなってくるでしょう。4人いると，50K÷12=4KB/packet，実効値ではその半分の2KB/packet程度しか使えないことになります。

2.LANの中でサーバあり

LANの中でサーバがある場合，入力の同期を取ることができます。いわゆるリフレクタといわれるサーバを用意し，入力をそれぞれリフレクタに送りあって，ひとつ前のフレームのデータをもらい，ゲームを進行することもできます。ピアトゥーピアだと3人，4人と増えるたびに，ゲームの世界を完全同期させることが難しくなりますが，リフレクタがいればその辺はクリアできます。しかし，リフレクタにデータを送ったり返したりする処理で，4人いればリフレクタは上りと下りで8つの接続をすることになります。したがって，10Mのときに20フレームのゲーム，1Syncおきに1パケットだとすると，お約束の500Kビット/packet，これに500K/8＝62kbpsで6KB/packet，実効値で半分程度の3KB/packetぐらいとなり，リフレクタがいることで，少しはバンド幅も稼げることがわかります。

リフレクタの場合，ネットワークの負荷はありますが，サーバのパフォーマンスが低くてもできます。もちろんサーバが落ちたらゲームはエンドですが，少し古めのLinuxマシンとかでも十分仕事をしてくれるはずです。サーバをLinux，ゲームのフロントエンドをWindowsで作るのは割と現実的な線かもしれません。Ultima Onlineのように，サーバのバンド幅と，なによりサーバの処理量がたくさんあるのなら，（たとえば，ちょっと古いLinux/Alphaマシンなら40万円ぐらいで探せば600MHzのマシンが買える）気合を入れて，サーバにゲーム処理のほとんどをさせてしまうって手もあります。サーバの負荷は重くなり，当然のようにゲームにアクション性は減ってきます。となるとゲームはアクション性が薄いことを考慮して，UOのような思考型にデザインしていくことも考えられます。

3.モデム直結でサーバなし

56kモデムは上りのみ高速なので，直結すると33.6kモデム相当になります。ISDNじゃなくアナログだとして，33.6kモデムで転送すると33600bpsの通信速度で，あまり圧縮を期待しないと3360B/sec通信速度（8ビットパリなしスタートビット1），決まりの20フレで1sync，1パケットを基本とした転送をする場合，3360/20packet＝168B/packetの転送ができます。これに加

えて，お互いの状況を伝えあって返す方法を取ると，この半分になります。

モデム直結の場合，すべて自分で管理できますから，パケットとはいえどもTCP/IPやIPXで作らずにお手製の転送で行えば，168バイトすべてを自分のプログラムで使えます。

ただし，そもそもの転送速度の遅さがあるので，入力を揃えて同期する方式だと，ネットワーク分の遅延がかかってしまいます。X-BANDなどのゲームをプレイしたことがある人は，知っていると思うのですが，それでもなんとかアクションゲームができる程度の速度にはなります。

一方で，モデムが全二重であることを生かして絶えずお互いでお互いの状況を送り続けるのであるなら，毎フレーム168バイトでデータを構成して，現状報告をし続けるというプログラムも作れます。あとはDiabloのようにお互いのマシンでお互いの当たり判定をするゲームならば，一応OKですよね。

4.モデムでインターネットを通してサーバなし

アナログの33.6kモデムを利用し，IP指定をしてお互いでピアトゥピア接続したことを考えます。Diabloが4人プレイまでサポートしている方式です。そして，これがおそらくいちばん難解です。大雑把に計算するといっても，ほとんどあてにならないでしょう。TCP/IPを使うということはヘッダが載るので，それだけで最少で56バイトは使われてしまいます。また，プロバイダなどとの応答形式によるので，ほとんど計算ができません。一応計算すると，理論値でプロバイダまで通信速度が168B/secだったとします。そのうちの56バイトを使われるので，だいたい1Syncあたり，100バイトぐらいしか転送できません。

こうなると，ほとんどお互いにお互いの入力を同期して行う方式は不可能に近くなってしまいます。

5.モデムでインターネットを通してサーバあり

よし，○○君のうちをサーバにしてゲームをやろう！　とするとさらに複雑になります。サーバには，すべてのプレイヤーのデータが送られるので，モデムのなけなしの20フレーム時の1Syncの1パケット，163バイトをシェアしないとなりません。ひとつにつき，56バイトのヘッダがあるので，もはやインターネットを通しては，秒間20パケットは不可能に近いでしょう。3人がぶら下がるなら，168/6＝28バイト。これではTCP/IPのヘッダも送れないので，1/4の転送量で済むように5フレームで送りあうと単純に，114バイト/packet。このうちヘッダに56バイトですから，実際のデータは約60バイト程度です。描画を5フレームにするとゲームにならないので，画面だけは20フレにして，いかにそれを補完していくのかが鍵となるでしょう。でも，この計算も理論値ですから，現実はというと……。

6.サーバのみOCNとか128kISDN

そして，上のケースでサーバだけがOCN128kで，クライアントにモデムをつなげてもらう場合，OCNのサーバの128kをクライアントでシェアすることになります。20フレなら，640B/packetのバンド幅があるので，3人ぶら下がるなら，約110B/packetの転送ができます。ヘッダを除くと50B/packet程度。あきらめて10フレで送るとこの2倍で100B/packet。ユーザーメイドでネットワークゲームを作る場合，投資の限界はこの程度でしょうから，この通信量でなんとかゲームが成立するようにゲームを構築しないとなりません。

もちろん，OCNは聞いてのとおり，常に理論値が出ているわけじゃないと考えると，これもフレームを落として，いろいろな工夫が必要になります。当然のようにモデム側も理論値が出るわけではありません。

■考察

コンピュータの性能が低い時期は，できること，できないことが限られすぎていて，ゲームを作るときのプログラマとプランナーが同じ人であることが多かったと思います。現在は，多少なりとも足かせが緩くなったせいもあって（もちろんゲームの種類にもよりますが），必ずしもプランナーにプログラマ的資質や，ハードウェアの知識が求められなくなってきました。コンピュータの性能を知らなくてもできる企画というのも現実的に存在します。

一方でプログラマ，どのポイントにいるプログラマかにもよるのですが，ハードウェアシステム内のバスバンド幅やリソースをいかに使い切るかを考えて，ゲームのシステムを構築していきます。

ネットワークというリソースは，ゲームの企画がシステムに左右されることが今後も多くあると思います。提供されたばかりで，ものがものである以上，これがある段階で足かせが緩くなったりして，あまり考えなくて済むようになるとはやっぱり思えません。いつの時代になっても，企画の根幹の部分までネットワークを意識しないとならないかもしれません。

クライアントが仮に10Mbpsぐらいの回線で接続される時代がきたときは，ゲーム用のサーバを利用するなら，サーバとインターネットを繋ぐ回線の太さ，サーバのパフォーマンスはいかほど用意しないと，破綻するのか？　というような様子でバス幅が上がれば上がるほど，いろんな部位での足かせが大きくなってくると思われます。

現時点においても，交流が可能なゲームのデザインは複数あると思われます。たとえば，IrDAでの無線通信でできることは限られてますし，モバイルPCと携帯電話でできることも限られています。Web上でJavaを利用したゲームであったとしても，制限を意識しないとなりません。低位レイヤーでネットワークを吸収し，信頼性を高く接続する方法論は，LAN限定で繋ぐことですが，それの場合やはり，隣人とでしか遊ぶことはできません。

インターネットを利用する場合は足かせはさらに大きくなることがわかりました。いつ切れるかわからないという信頼性の低さもあります。もし，ユーザーメイド（同人とか）でゲームを作るなら，サーバを用意するのかしないのか？　サーバを用意するならば，どのぐらいのパフォーマンスがいるのか？　たとえばテレホーダイ時間を利用したISDN接続のPCをサーバにしてもシステムのパフォーマンスやバンド幅 は足りるのか？　そのとき，サーバに求められるバス幅，CPU性能はいかほどなのかを，ある程度予測できなくてはなりません。

結局のところ，転送しあうことができるデータは微々たるものであり，それがいつ切断されてもおかしくない状態です。そういう通信状態でゲームを制作する際，低位のレイヤーでネットワークを隠蔽することは難しくなります。ではどうやって作るのかといえば，アプリケーションレベルのレイヤー，企画レベル，ゲームのルールのレベルで考えねばなりません。たとえば，ゲームに参加している人の誰かひとりが回線切断しても，ゲームが破綻しないように作るなり，1人ひとりの通信の速度が違ってもゲームの根幹にあまり影響しないように作るなり，切断，再接続を繰り返しても問題ないようにすることが必要です。

そう考えてみると，これほどいろんな問題があるにもかかわらず，DiabloやUltima Onlineはこれらすべてのことをうまくやっています。先のほうで，ネットワークでゲームを作るには，2つのゲームプログラムを接続しなくてはならないというテーマを挙げました。これが，〝ドグマ〟であるといったのは，世界をきっちり接続できるほどネットワークは信頼性が高くなく，また速くもないからです。Diabloのように，プレイヤーの見た目上だけの世界が同一であるなら，細かいものは同一でなくてもいいわけです。つまり，企画レベルでそこまで厳重にパラメータが同一であるかどうかが必要ないように作られているあたりが，あのゲームの秀逸なところでしょう。

結局，スケーラブルで信頼性が低い回線を通る際に問題になるのは，いかに人間，プレイヤーの目から見て同じように見えるのかがキモになり，そう考えたとき，ゲームというルールの層までも砕ききって，うまく作り上げないとならないという状況です。もちろん，非リアルタイムにする方法論もありますし，ゲームによっては大半が予測で計算できるものもあります。どのぐらいが割り切りどころかが勝負です。

CPUの速度もそうでしたが，ネットワークも面白いもので，比例的に速度が変わっていっても，できることは比例的に広がるわけではありません。10MBASE時代では考えられなかったことが，100Mではできるようになりました。1GBASEあれば，CPUパワーの切り売りも，いろんなところでできるようになってくるでしょう。ますます，クライアント側からの見た目上で，この処理がどのマシンで行われているかわからない状態が出てきて，それでもよい状態になってくるかもしれません。

作る側からすれば，どんどんお勉強が必要な時代になってくることは間違いないんですよね。

ネットワークゲームの地平へ

インターネットセキュリティ事件簿

三沢和彦 *Misawa Kazuhiko*

最近ではOCNやODNエコノミーなどを使って個人でも常時接続のサーバを立ち上げることもできるようになってきました。SOHOなどで，個人経営のWebサイトなども増えています。その気になれば，個人でゲームサーバを立ち上げることも不可能ではありません。しかし，気をつけなければならないのはセキュリティの問題です。ここでは某大学で現実にあった事例をもとにインターネットのセキュリティについて考えてみましょう。

■ケース1：スパムメール

「メールスプーラがあふれているぞ！」

私が異常に気づいたのは，我々のメールホストがFile system fullの警告を発したときであった。それとほぼ時を同じくして，私のオフィスの電話が鳴った。

「そちらのメールホストからダイレクトメールがばらまかれているっていう苦情が寄せられているぞ」

電話の声は，我々のホストが属するドメインの管理者であった。

「しまった」

私には，思い当たることがあった。我々のメールホストには，いわゆるスパム対策が施されていなかったのだ。いや，なにかある前に対策を講じよう，と思っていた矢先のことであった。

大量のメールのためハードディスクがパンク状態でいまにも倒れそうなホストであったが，かろうじて，ディスクの中身を見ることができた。案の定，root宛のエラーメールがスプールを埋め尽くしていた。エラーメールの中身を見ると，宛先のホストは現実に存在するホスト名が記されていた。世界最大のソフトウェアメーカーや，世界最大のインターネットプロバイダのアドレスもあった。しかし，宛先ユーザー名はランダムな文字列になっている。しかも，差出人は我々のホストにもない架空のユーザーになっていた。つまり，何者かが我々のホストのユーザーのふりをして，世界中の架空アドレスにいたずらメールをばらまいたということになる。宛先が架空であるため，山のようなUnknown userのリターンメールが我々のホストを襲ったのだ。

宛先アドレスは，架空アドレスだけではなかったようだ。一部は実際のユーザーに無事(？)送り届けられ，それがいたずらであることがわかると苦情のメールをドメインマスター宛に送ってきたのだ。そして，その苦情を受けたドメインマスターが私のところに電話してきたわけである。

このケースは，スパムメールというものである。このスパムメールとはどういうものかを解説するために，まずインターネットメールの初歩的な仕組みを説明しておこう。

インターネットでは，一般にSMTP(Simple Mail Transfer Protocol)サーバというメールを送受信するサービスが用いられている。メールを送信する場合，メールの本文に，発信元，宛先，日付などのヘッダと呼ばれるメール情報をつけて，SMTPサーバ宛に送る。メールデータを受け取った発信側のSMTPサーバは，そのデータをヘッダと本文に分けて，スプールディレクトリというところに一時的に格納する。この操作をメールキューという。

送信側SMTPサーバはヘッダを解析して，宛先アドレスから受信ホストを見つけ，その受信ホストと交信を開始する。この交信のプロトコルがSMTPである。受信ホストに宛先アドレスのユーザーが存在すれば，ヘッダ情報とともに本文を受信ホストに送り，受け取った受信ホストはそのメールをユーザースプールディレクトリに保存する。受信者は自分のスプールにメールがきていれば，それを読み出すというわけである。

さて，ここで，最大の問題は，「SMTPサーバは，自らのホストに登録されたユーザーでなくても，まったくの第三者のメールでも発信することができる」という点である。これをメールの中継という。すなわち，外部の人間が自分のところのメールサーバに送信メールデータを送りつければ，それを自動的にほかに送信してしまうのである。この点がスパムメールを理解するためのミソである。

今回の場合，我々のホストから送り出されたダイレクトメールはヨーロッパのある国のホストから送られてきたものであることがエラーメールのヘッダ情報を読むことによってわかった。つまり，イタリアのホスト上の犯人がランダムなユーザー名を著名なサイトのアドレスに適当にくっつけて何百もの宛先リストをでっち上げ，これらのダイレクトメールを自分のホストから発信せずに，我々のホストに中継させて全世界にばらまいたというわけである。

最終的に発信したホストは我々のホストであったので，エラーメールは送信ホストに集中してきたため，メールサーバがパンクさせられる羽目となったのだ。

先ほど，「思い当たること」というのは，我々のホストは現状では不特定多数の外部ユーザーにメールの中継を許可する設定のままにしてあったということである。おそらく，いまだに多くのホストはメール中継の設定に注意を払っていないかもしれない。私自身，早く対策せねばと思っていたが，我々のホストで走っていたSMTPエージェントのsendmailはバージョンが古く，標準でメール中継のセキュリティ対策をサポートしたバージョン8.8がリリースされたばかりであった。とはいえ，これも自分の怠慢を弁解する理由にはならない。

最近では，UNIXに限らず，WindowsやMacintosh上でも，使い勝手のよいメールサーバソフトが普及してきている。どれも皆いま述べたメール中継に関する制限が加えられるようになっている。とりあえずは，外部の第三者が勝手にメールを発信できないように設定しよう。さもないと，次のようなことも起こりうるかもしれない(この部分の内容はまったくのフィクションです)。

●架空事例

あなたは，最近はやりのSOHO(Small Office Home Office)事業者である。インターネットビジネスには，個人サーバも必要なので，自分でドメインを取得し，自宅に常時接続線を引いて，サーバを自分で管理することにした。メールアドレスも自分独自のドメイン名であり，これで宣伝効果も期待できると思っていた。この業界は競争も激しく，リピータのクライアントを確保するのも，大変なことであったが，どうにか努力して事業もやっと軌道に乗せることができた。同業者とも，ライバル同士でもあるが，それなりの信頼関係を築けたと思う。

あるとき，あなたのメールサーバから，あなたの同業者を誹謗中傷するメールがばらまかれた。

「A社もB社も，仕事の質は最低だ。そのくせ，相場以上のコストを要求している。我が社は，ほかと違って良心的なサービスに自信を持っている。他社に依頼するのは，時間と金の無駄だ！」

ヘッダも操作されていて，あなた自身が送信し

たことになっている。あなたのところには，苦情のメールが殺到した。
「ふざけるな！ 正々堂々と仕事の質で勝負しろ！」
挙句の果てに，架空メールの中継をさせられて，あなたのメールサーバはダウンしてしまった。今度は苦情の電話が止まらなくなった。
あなたの個人サーバから発信したものは，どのサイトにも拒否されるようになってしまった。もはや，直接会話して謝罪しても失った信用は回復しなかった。あとから風の噂で聞いたことだが，あなたのライバルがあなたを陥れるためにでたらめメールを仕掛けた結果らしかった。

■ケース2：ポートスキャンアタック

事件はある大国の軍からの電話で始まった。我々が管理するドメインのあるホストより ポートスキャン（Port Scan）がかかっているので，至急調査してほしい，という問い合わせであった。我々の計算機センターになんと日本語の通訳つきで問い合わせてきたのだった。
「国際問題に発展する……」
我々の間に緊張が走った。何者かがインターネットを通じて，我々のドメインのホストから某国軍に攻撃を仕掛けているというのだ。
ここで，ポートスキャンについて説明しておこう。インターネット上でTELNETやFTP，SMTPなどのサービスを使用するときには，相手ホストのサービスポートに通信を試みることから始まる。ポートにはそれぞれのサービスに対応したポート番号が割り振られている。たとえば，TELNETは23番，FTPは21番，SMTPは25番といった具合である。
外部からの不正侵入者は，攻撃対象のホストにこれらのポートにしらみつぶしに通信を仕掛けることによって，通信可能に設定されているポートを探し出す。侵入者の目的はアカウントの乗っ取りであり，まったく関係のないホストを自由に操るためのカモアカウントを確保することである。そのホストのroot権限を乗っ取れば，正規の管理者と同様にそのホストの設定を自由に変えることができる。それどころか，rootのパスワードを変えてしまえば，正規の管理者さえもログインできなくなってしまうことになる。
しかも，利口な侵入者なら，本来のユーザーに気づかれないように，こっそりと悪用する。そのためには，アクセスのログなどを改竄して，不正使用の足跡をできるだけ消しておこうとするものである。さらには，ポートスキャン行為そのものを相手に悟られないように行うことができるのである。
では，アクセス可能なポートを探し出すことがアカウントを乗っ取ることとどのような関係にあるのだろうか？ とにかく，パスワードファイルを盗んだり，あるいはOSのバグをついて，パスワードを知らなくてもroot権限を行使できる状況を作ったりするのだが，それには，相手ホストの内部でなんらかの操作を行わなければならない。インターネットを通じてリモートでやり取りするには，このポートだけが唯一の通り道なのである。しかも，単に通信可能なポートを探し出すだけでなく，セキュリティホールと呼ばれる，不正使用の役に立つバグのあるポートを探し出すためのツールなどもアンダーグラウンドには出回っている。
我々はさっそく調査を開始した。その結果，わかったことを時間経過に沿って報告しよう。

0. 某年8月から9月にかけて，我々のドメインのホストが外部からポートスキャンを受けた可能性がある。これにより，不法侵入できるホストが洗い出されてしまった。

1. 同時期，某研究所でスニファ（snifferパケット盗聴プログラム）が仕掛けられ，複数のパスワードが盗難される。その研究所は我々と仕事上のつながりがあり，頻繁にアクセスがある。したがって，その研究所のホストから我々のサイトのホストにtelnetでログインした場合は，我々のサイトにあるホストのユーザーパスワードが盗まれることになる。

2. 10月3日早朝，あるホストが外部から侵入を受けroot権限を盗まれた。

3. 侵入者は得たroot権限を使い，世界各地のサイトへポートスキャンを行った。世界中で少なくとも数千，おそらく数万にも及ぶ数のホストに対してポートスキャンを行ったものと思われる

4. 10月3日朝6時30分，ポートスキャンを受けたサイトから，電話で連絡がきた。攻撃された側から見ると，我々のサイトの内部にいる人間が実際に攻撃しているのか，我々のホストが単に利用されただけなのかは区別がつかないのだ。

5. 同日午前11時，そのホストの管理責任者に連絡を取ることを試みるが残念ながらつかまらなかった。

6. 同日夜21時30分，ポートスキャンプログラムが走らされていたマシンを探し出し，ネットワークから切り離す。

7. 乗っ取られたホストに残っていたログの解析など，被害状況の調査を行う。侵入者が仕掛けたポートスキャンプログラムのソースコードが見つかる。

8. そのときに発見されたポートスキャンプログラムを使って，再度，ドメインのマシン全体に対して，ポートスキャンを行い，侵入者がすでにセキュリティホールを入手したと思われるセキュリティ未対策のマシンをリストアップする。

9. 問題点と対策を公表し，対処を徹底した。

我々は緊急に対策を講じる必要があった。なぜなら，このような状況を放置すると，我々のドメインから送られてきた情報はすべて拒絶されてしまうことになってしまうからだ。
侵入を受けたマシンに関しては，踏台などにされることにより，ほかへ被害を与えることがきわめて深刻な問題になるのだ。ネットワークに接続するということは，常にそのような危険にさらされていることをよほど肝に銘じておかなければならない。

■セキュリティ対策の実際

外部に被害を拡大させるケースはだいたい次のようなことが想定される。

1）スパムメール
これはケース1で取り上げた。でたらめメールを大量に中継してしまうことになりかねない。対策は第三者からのメールの中継を制限する設定を加えること。

2）スキャナプログラムの実行
インターネットに接続されているホストのセキュリティ上の弱点を探る。弱点を突かれて侵入されると最悪の場合，ホストのroot権限が乗っ取られ，今回のように情報犯罪の舞台にされてしまう。対策は不必要なネットワークサービスは停止して，さらにアクセス管理ユーティリティを用いることで，ホストに対するアクセスを厳しくチェックする。

3）スニファの実行
スニファプログラムを仕掛けられたマシンが存在するネットワーク上のマシンとパスワード照合のあるやり取りを行うとパスワードが盗まれる。この結果，リモートで利用しているシステムが芋づる式にセキュリティを破られてしまう危険性が高い。対策にはパスワード照合に暗号化パスワードを用いる。

今回の事件では，ポートスキャンチェック以上の被害は報告されなかった。しかしながら，我々のホストを乗っ取った犯人が，他国の軍事機密を盗み出そうとしたり，銀行口座の残高を勝手に書き換えたりしようものなら，本当に社会問題・国際問題に発展するところだった。
現在，爆発的人気のPC-UNIXではあるが，その導入はきわめて簡単にできる半面，管理コストは大きい。特にLinuxなどはOSのソースが公開されているために，バグも非常によく研究されている。侵入者は常にあなたのホストを狙っているのである……。

SPECIAL

特集 ネットワークゲームの地平へ

インターネットゲーム時代のコミュニケーション(?)論

古村 聡 *Komura Satoshi*

最近ではネットワークといえば当然インターネットのことを指します。この匿名性の高いメディアには独特のコミュニケーションの取り方があるようです。「ネットワーク経由だから」気をつけなければならない問題について実例を交えつつ検討してみましょう。

インターネットを有効に使っている人気ゲームと，そりゃぁなんといってもUO(ウルティマオンライン)ですけど，実をいうと私はプレイしていなかったりします。話を聞いているとそりゃぁ面白そうなゲームですし，確実にハマるだろうなーと。時間も通信費用もガンガン使ってしまって，家計費は底を尽き，子供は泣き，家庭崩壊……それを考えるととても恐ろしくて手がつけられないのであります。……昔風にいうと，親の遺言級ゲーム(ああ，懐かしい)だからという奴ですね。

で，アーケードゲームが代わり，というのもなんですけど，最近ハマっているのがコナミのDance Dance Revolutionというゲームなのであります。ゲームセンターで人だかりがしているので内容をいえば，「ああ，あれかぁ」と納得していただける方も多いことでしょう。

ミラーボールこそないけど，そこは黄色や紫のフラッシュが光り，デーンと2人分のお立ち台がそびえる妖しいゲーセンのなかでも，さらに妖しい空間。

で，ここでズンズンと響き渡るサウンド(70's なのよこれが。Butterflyとか)と画面に現れる表示にあわせてプレイヤーはステップを踏み，このステップが正しく踏めるかどうかでスコアが決まるというゲームなのであります。

これで踏んで踏んで踏んで踏んで踏んで何十回というプレイに費やしたお金と引き換えにHARDランク(といってもここから下よりも，上を見たほうがランクがたくさんあるんですが)のButterflyがちゃんと踊れ……もとい踏めるようになりました。大馬鹿もんですな。まぁ，子供といっしょに風呂に入る家族サービスは欠かさないので家庭崩壊の危機はちゃんと免れていますが，次はHARDレベルのPARANOIAクリアして，いよいよDDRでいちばんカッコイイといわれるANOTHERレベルに挑戦……さぁ，そのときも私，(で)は家庭を持っていられるんでしょうか？ (そこそこ，妙な期待しないよーに)

■みんなの意識がひとつになって

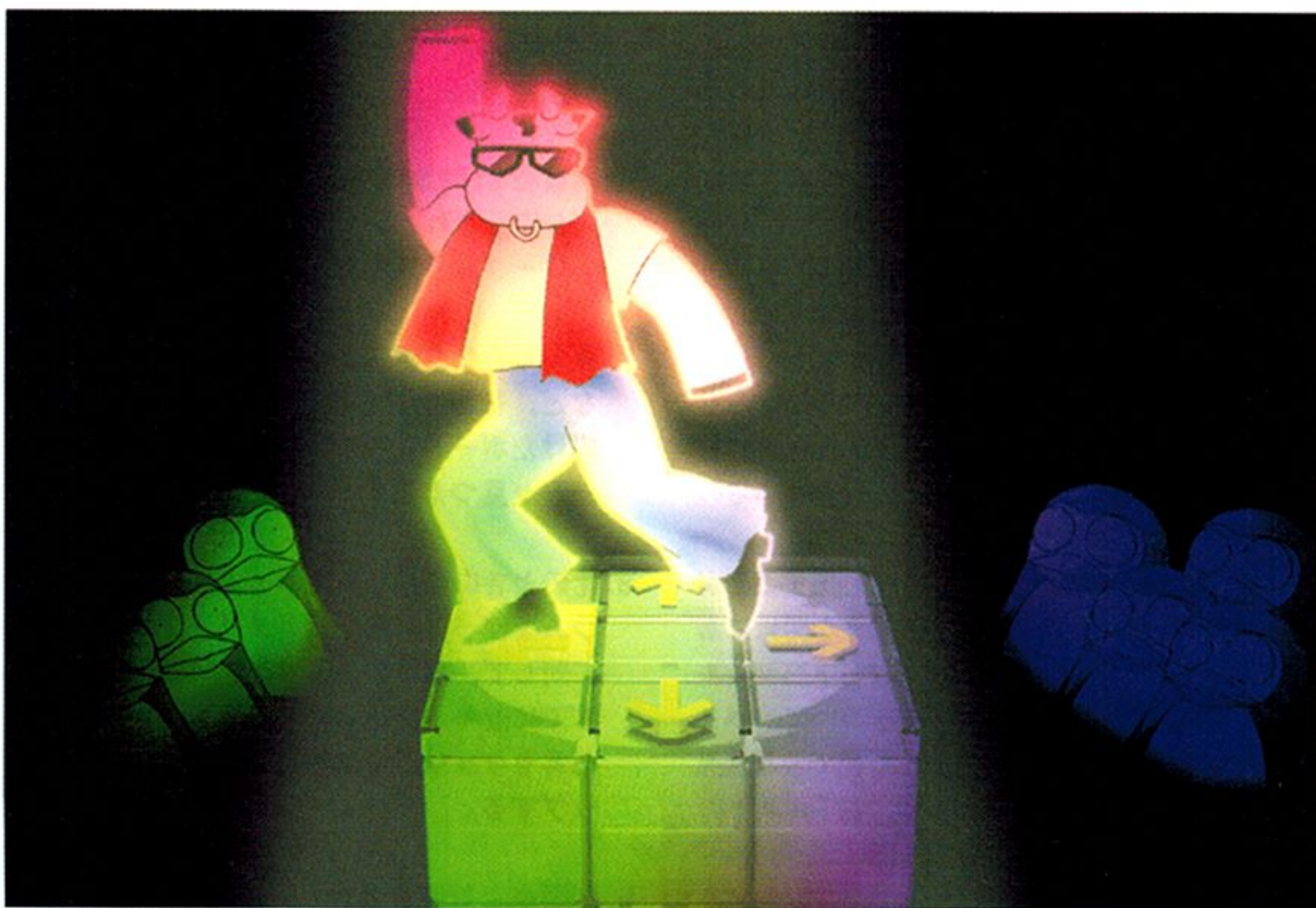
このDance Dance Revolution，略してDDRはこの文だけでもわかるように非常に恥ずかしいゲームです。いや，ゲーム画面の70'sなアフロにーちゃんとか雰囲気自体も恥ずかしいんだけど(ベルボトムとかも復活しているし，若いもんにはこれがいい感じなんでしょーか？　わからん)，それ以上に恥ずいのはプレイしている本人です。だって，自分は画面のほう，つまり観衆(いる，っていうかたいてい群がるという表現が正しいくらいいるんですね，少なくとも私のプレイするような時間には。まさしくギャラリーという感じでステージ……じゃなくてゲームを取り囲む人々)な方々には背を向けているとはいえ，見られているんですよ。見えなくたって背中に視線が痛い。ああ。

踏めればいいです，カッコ悪くたってちゃんと最後まで踏めればいいじゃないかと頭の中でいい聞かせているのに，ああ，もしかして自分の踊りはカッコ悪いのじゃないか，足だけで踏んでるんじゃないか，上半身が固まってるんじゃないか，むやみに力まかせで踏んでいるんじゃないか，実は全然リズミカルじゃないんじゃないか，楽しそうに見えないんじゃないか，とかそんな考えがぐるぐる頭をよぎったりして，汗をかきながらゲームを終わってみると背後はいつの間にか1人になって，しかもその人は次にプレイしたいだけだったりしてああ自己嫌悪。

とにかく，観衆を意識してしまうゲームで，カッコよくないとなんか申し訳ないような，そんな気分になってしまうのであります。

そんなに恥ずかしいんだったら，やめてまえばいいではないか，と思うんですが，このゲームには不思議な力があります。

いままでもギャラリーに見られて結構恥ずかしいゲームってのはいくつかありましたが(X68000全盛のころだとギャラクシーフォースとか……懐かしいなぁ)，このゲームのちょっと違うところは，ゲームをプレイしている人たちの「場」のようなものがあること。

ボサ髪
真っ赤なバンダナ
銀縁メガネ
皮ベスト

ケミウォッシュなスリムジーンズ
「あっぱれ」と書かれた扇子

という，ポイントはかっこいいんだが，よくわからないカッコ（まぁ，飾りをつけたオタクか……）で踊ってる方もいらっしゃったりするのも，きっと田園都市線沿線某ゲームセンターだけの現象ではありますまい。

しかもこの方，ジーンズのポケットに片手を突っ込んだまま扇子を手に，固まった表情でくるくる回る，という。そう，なにしろ正しい位置さえ踏めればいいのですから，譜面を覚えているなら体が画面を向こうが，ギャラリーを向こうが構わないわけで……ゲーム中にクルッと回ってみせるのもカッコイイDDRのポイントなのであります。でもゲームの間中ずっとぐるぐる回ればカッコイイというものでは……。

いや，黒コートでダンダンとANOTHERのPARANOIAを踊るように踏んで（いや，踊る，でいいんだけど），いくソツのないミスチルの桜井モドキなカッコイイ少年もくるんですけどね。やっぱり，なるんだったら，飾りおたくではなくて，ああいうカッコよく踊れるようになりたいなぁ（もう少年じゃないけど），と思ってしまうのですね。いや，私だけが思っているのではなくて，お互いに「む，ああいうふうにカッコよく踊りたい」「あいつよりはカッコよく踊りたい」「とにかくカッコよく踊りたい」と，その結果はどうあれ，プレイヤーに「カッコよくなりたい」という共通意識とともに，どうもプレイヤー同士がお互いに意識しあった，なんともイっている感じがするんですよね，どうも。

その証拠にゲーム台から降りるとときどき目があってしまったり，すごいカッコいい人がいるとお互いに目を見合わせてしまったりもします（実体験としてANOTHER MIRROR DOUBLEという最強の組み合わせで，恐ろしくカッコよく踊っている人を見て，見知らぬ隣人に「すごいっすねぇ…」と同意を求められてしまったことが私にはあります。確かにあれは，見知らぬ人にも思わず同意を求めたくなるくらい凄かった……です）。男は拳で語る，じゃなくて足で語っているのかもしれません。

しかも，ゲームセンターに同じ時間帯に集まる人たちは，いつも同じ人が何人かはいるので，なんというか，ゲームがクリアすべき「目標」であると同時に，みんなが集う「コミュニティ」になりつつあるような感じさえします。あくまで声はかけないんですけどね。

どうでもいい話ですが，やっぱりこのダンスがいいのか，サイケ感が受けるのか，単に楽しそうに見えるのかよくわかりませんが，女子高生とおぼしき（って制服なんだから間違いなくそうなんだろうけど）女の子たちや，それより年上の女の子たちも踊りにいらっしゃいます。先の「飾りつきおたく」君たちは，この子たちには足でなく口で語っていた（つまりはナンパしてたみたい）なんですが……まぁ，がんばってくださいね，若人たち（おじさんはかわいい妻と子供が家で待っているのでそそくさと帰ります）。

■「場」は同じ目的を持った者の集まりである

って，ネットワークとあまり関係ないものばかりを語るのもアレですので，ちったぁネットワークなゲームの話もしましょう。

スタンドアロンのゲームと違って，インターネット上での対戦ゲームというのもまさしくこの「無言のうちにひとつの目的を持った者たちの集まり」として考えると成功するゲームのジャンルでしょうね。

そう，ウルティマオンラインをやらない私ですが，ネットワークゲームをまったくプレイしないというわけではないんです。というか，逆。インターネット麻雀「東風荘」にはまりすぎてしまって電話代と課金に懲りた（ん？　テレホーダイ？……残念ながら，夜11時以降に行っても東風荘が満員で入れないんです）。

経験上，自分で自分に封印をしているのですよ。そんな社会人の財力をもってすれば……電話料金はともかくとして，プロバイダの課金が3万円とか4万円とかいくと，パパおこづかいなくなって，駅で缶コーヒーも飲めなくなっちゃうのですよ。ああ，情けない。もう何カ月，お酒なんか飲みに行ってないことか。

ともかく東風荘。ここは簡単にいうと，最大512人のプレイヤーが入ることができるインターネット雀荘でして，見知らぬ（別に見知っていても構わないのは普通の雀荘と同じです）4人ずつが適当に卓に参加して麻雀をするソフトです。で，ゲーム進行中にチャットもできる……というのがウリだったりもします。

で，私（で）は，今回の特集がネットワーク対戦ゲームであると聞いて，「それじゃぁ，東風荘で卓を囲んで，そのときのチャットのログをここに載せれば簡単にページが埋まってラッキー」と思ったのですよね。ではここにそのログを公開しましょう。

（ゲームが始まる）
Player1：ども
で：どもー
Player2：どもども
Player3：よろ
（（で）が振り込む）
Player1：あり（←ありがとうの意味）
で：ああっ，またやった（^^;
Player2：おめでとー
Player3：おめ
（Player1ツモる）
Player1：すま。つも
で：おめっとー
Player2：おめ
Player3：おめ
（以下略）

という程度の会話しかしてないので，なんか，

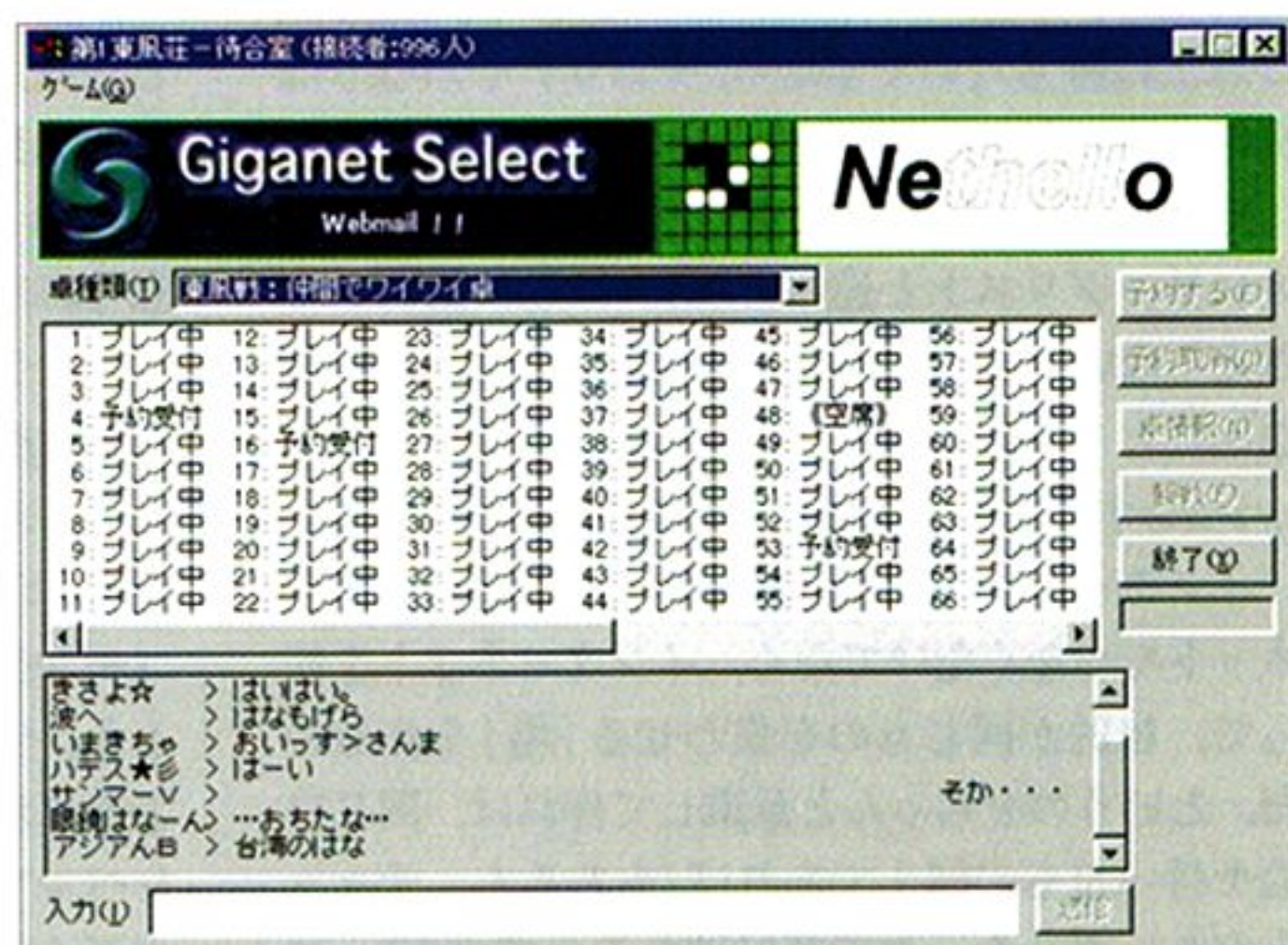



図1　ずらっと卓が並んでいる

図2　コミュニケーションは簡略化されている

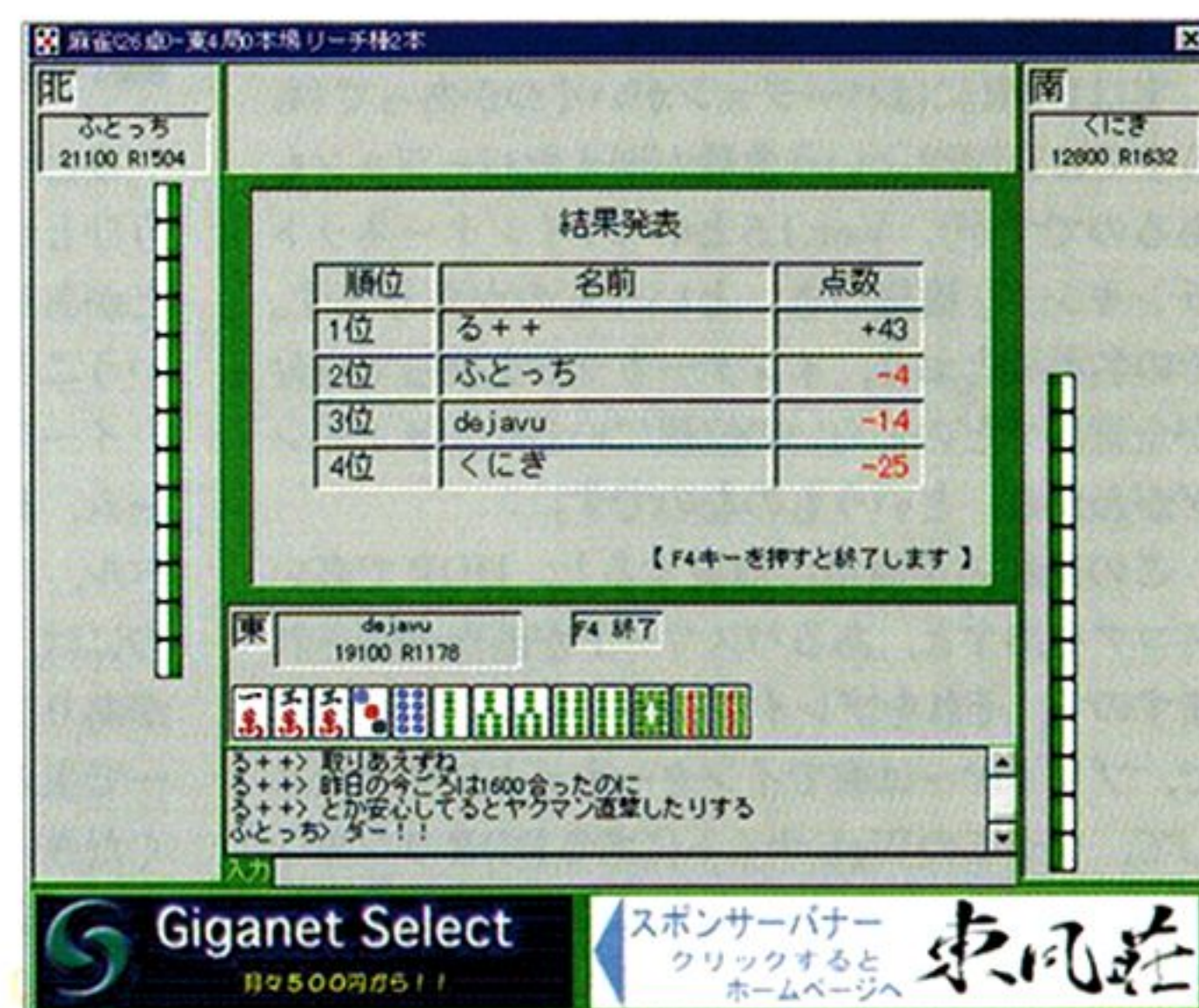



図3　うーむ，いまいち

実はほとんどチャットには意味がないのでありました。あと，してもたまに，どこからアクセスしてるかと，天気の話が出る程度なんですよね。「福岡は雪降ってます」「新潟です。雪で電車止まりました」「私はサイパンの事務所から」。それ自体で盛り上がるというようなものではないのですね。

まぁ，考えてみれば，負ければ自分のレート(東風荘では自分のデータとして1500点を基準として入会してからトータルでどれだけ勝っているか負けているかを示す「レート」というものがあってRいくつ，という数字でゲーム中は常にハンドル名とともに表示されるのです）が下がってゲーム中それが表示されてしまうわけですし，卓を囲む以上は勝ちたいですから，やはりチャットよりは真剣に牌に向かうのが当たり前，というものです。

そう考えると同じインターネットでも掲示板やメーリングリストと違って(あれは逆に読書家でさえしないような字数の文字を読まなければならないこともあるので)，言葉を出してコミュニケーションする必要のあるものではないのですよね。

インターネット対戦は単にユーザーをインターネットでつなぐだけでなく，インターネットを使って，目的が同じものを集わせる「場」を作るのだ，というのをちゃんと意識して作れば，同じ目的を持ったもの同士であれば(もちろん，チャットがあったらあったで楽にはなるのですが)，コミュニケーションに言葉はそれほど必要ではないのです。麻雀の場合は捨てる牌の1つひとつが言葉なのですから，これでいいのです(ああ，すると私は麻雀でも馬鹿なことばかりいっていることに……自慢じゃないが役満放銃したことはあっても自分が役満どころかトップを取るのもめったにないR1242のタコです)。「ネット対戦はひとつの目的を持った者たちが集う場所。余計な言葉は(あまり)いらない」なのです。

■「場」はルールから逸れてはいけない

ところでやっぱり話がDDRに戻ってしまうんですが，実はこいつもインターネットを使って「場」を提供しています。といっても，インターネットを使ってゲームができるわけではないので，いわゆる「ネットワーク対戦」とはちょっと意味が違うんですが。

実はDDRにはバージョンがいくつかあって(もうすでにDDR2という曲数が増えたバージョンもあるのですが)，Ver.1.5という「インターネットランキング」機能つき，というものがあります。その名前のとおり，インターネットを使って自分が全世界でどのくらいの位置にいるのかランキングがわかる，というものなのです。

この仕組みを簡単に解説すると，DDRで高いスコアを出すと，あるパスワードが画面表示されますので，それをプレイヤーがメモしていきます。で，プレイヤーは家でインターネットにアクセスして，コナミのWebサイトにあるDDRランキングページでそのパスワードを入力すると，あなたの世界中での順位がわかる，というもの。まぁ，ゲームセンターでインターネットに接続できるわけでもないのでこういう仕組みになっているのだと思いますが。

でも……これが，私見ですけど，DDRの盛り上がりの割にはあまり盛り上がっていないように見えるんですね。少なくとも，私は，ゲームが終わったあとでちゃんとパスワードをメモっているプレイヤーというのを見たことがないです(え，私？　私はその，まだ，HARDレベルなので……ハナからパスワードなど出ません。でも，出たところできっとメモらないだろうと思われます)。

思うんですが，これの失敗(にしてしまおう，この際)の原因って「あまりカッコよくない」ことにあるのじゃないかと思うのです。先にも述べたとおり，DDRに踊りにくる人ってハイスコアを出しに踊りにくるんじゃなくて，その人なりに「よりカッコよく」踊るためにくるんじゃないかと思うわけですよ。

で，ゲームが終わったあとにおもむろにメモ帳を取り出して，メモを取るというのは……なんかこう，未練がましい感じがしてカッコよくないんですよねぇ。やっぱりゲームが終わったらパッとコートを翻して立ち去るのが粋！　ってもんでしょう。少なくとも，このゲームにおいては疾風のように現れて，ANOTHER DOUBLE(難しいレベルで2人分のパネルをひとりで踏むモード。すごく難しくて，かつ100円で1曲しかプレイできない，ヘタクソかつ貧乏性人間(俺だ俺)には絶対手が出せないモード)を1プレイだけして，さっと去っていく，くうーっ，これこそ粋！　漢と書いてオトコと読む！　というものでありまして，ゲームが終わったあとにメモを取るなんざ(皆さん早く次のゲームがしたくて待っているのに)，「やだ，あいつ，メモってるダサダサ～」とブレザー，チェックのミニスカの女子高生2人組に後ろ指さされること必至なのであります。

インターネットにみんなが集まれる「場」を作ろうということ自体は決して間違いではないと思うんですね。でも，「カッコよさ」を求めるゲームに「カッコよくない」プロセスを踏ませてはいけない。インターネットを皆が集まる場にしようと思ったらなにを目的とした人たちがそこに集まるのか，それをよく考えつつ設計しなくてはいけません。

■「場」は不安も増幅する

ひとつの目的を目指す者を場に集めるには，もうひとつ必要なんじゃないかと私は思っていることがあります。それは「ルールは単純化すべし」ということ。

イースをはじめとしたRPG，アドベンチャーゲーム，最近ではToHeartのようなビジュアルノベル，18禁ゲームにいたるまで，ゲームというものには，プレイを進めるためのルールというものがありますね。「ゲームを進めるためにはテンキーで主人公を操作して敵を倒して経験値を上げてください」「マウスで選びたい行動を選んで，女の子と仲よくなってください」とか。

で，これをプレイした結果「そして混沌とした世界が再び2人の女神とともに平和に戻るまでの感動のストーリー」を見ることができたり「あるいはマルチとの感動の再会へのストーリー」を見ることができたわけです(ああ，マルチマルチマルチ～。ごろごろ)。

で，得られるものは作り手によって「あらかじめ隠されていた」ものなわけですが，思うに，対戦型ゲーム以前のゲームプレイヤーの動機には「公開されたルールの裏に作者が隠したもの(ストーリーとか，アクションゲームならテクニックとか)を探す」という部分が大きな比重を占めていたように思います。

ほら，あなたもゲームをプレイするのに「この女の子たちとどんなストーリーが待っているのかなぁ，わくわく」とか「今日はうまくキャンセル技から12ヒットコンボにつなぐ技を見つけるのだ」とか思ったことはないですか？　そして，ゲームの作り手もそれに応えようと，さまざまなものをゲームのルールの裏に詰め込もうとした結果，いままではゲームのルール(ゲームをクリアするための暗黙のルールも含めて，たとえばフラグの立て方とか)というのを作り手も「次回は変えたほうがいい」と思ってしまう傾向があったように思うのですね。1，2，3……と続いているゲームを思い浮かべてください。システムやゲームの展開がまるっきり同じゲームってほとんどないのではないかと思うのです。

これまではそれでもそれほど問題はなかったんです。多少ルールが複雑になっても，プレイヤーが時間をかけて慣れればいいだけの問題だったんですから(最悪「いいや，こんなクソゲー」とポイしてしまえば)。

ですが，「場」に人が集うタイプのゲームでは，これは致命的です。このタイプのゲームでは，ルールはとにかく人に簡単に理解されるをもってよし，とされなければいけません。そのゲームが新しいものであればとにかく理解しやすいように単純なものを。続編のようなゲームであれば，そのゲームをプレイした人の暗黙からまったく外れないルールにすべきです。

たとえば，最近のゲームといえば，特にインターネットを使った対戦型ゲームです(ああ，よかった，やっと特集テーマ「ネット対戦ゲーム」になった)。

たとえば，インターネットを使った「信長の野望インターネット」。もはや説明の必要もない「信長の野望」をインターネットでセンターに接続して多人数＋コンピュータで対戦をする，というものなのですが，このゲームの体験版がいくつかのパソコン誌の付録CD-ROMなどについていたんですね。これを使えば(昨年一杯で体験版試用期間は終わってしまいましたが)，ゲーム中の1年分を体験プレイできたのです。

で，私も何度かプレイしたのですが……なにかプレイヤーが楽しんでプレイしているという感じが伝わらない(というか，ゲームにとまどっているとか，プレイがつまらない，という感じが伝わってきてしまって)，どうも，面白いとまで思えなかったんですね。

問題の原因として思いつくのはこの信長の野望

インターネットではゲームルールが多少，これまでの信長と違ってしまっていることです。信長というと「街を作って，農地を開墾して，豊かになったら軍を揃えて，さぁ，それから戦争だ！」という先入観が私にはあったのですが（というか，それが一度でも信長をプレイしたことがある人の先入観だと思うんですが），まず，それが「期間はゲーム中の1年」という体験版ルールがあるため，とにかく早く攻め込まなくては……と思ってしまう点にあります。

つまり，ゲームの進め方自体をどうしたらいいものかとまどってしまい，さっさと借金してでも軍を強化して攻め込んだほうがいいのだろうか……と。敵国も割と早い時期に攻めてきますしね。

さらに，問題なのは，ゲームのシステム自体にいろいろいままでの信長と違う部分がある点です。たとえば，「信長の野望」なんだから，隣国に戦争をしかければすぐ戦争になるのが普通だと思うんですが，このインターネット版ではそうではないのです。領地自体がマス目のようになっていて，戦争を始めるにも「兵の自国内での移動」という概念が入ってしまいます。初めて見たときにはこれをどうしたらいいのかよくわかりません。

これらは別にインターネット対戦でさえなければ，別に許容範囲のルール変更だったのでしょう。それどころか，スタンドアロンであれば，システムに慣れるための作業自体がゲームとしては面白かったかもしれません。ところが，ネット対戦プレイでは，たとえば8人のプレイヤーのうち何人かがルールに不案内でゲームにまごついてしまうのです。ゲームがスムーズに進みません。

しかも，ネット対戦の恐ろしい点はそれが「場」である，ということです。インターネット接続，特にダイヤルアップ接続では「抜ける」ということが簡単にできてしまいますが（「落ちる」という表現のほうが一般的な気もしますが，いきなり接続を切ることです），そこでひとりが抜けるとそれがほかのプレイヤーたちにもわかってしまいます。抜けたプレイヤーの「つまらなかった」という無言の意思表示がそこでほかのプレイヤーみんなに伝わってしまうのです。

しかも，相手はインターネット上のプレイヤーです。そのうち何人かは，掲示板やホームページで「このゲームはつまらん」という書き込みをするかもしれません。

信長の野望はそこまでいってはいませんが，最悪，インターネット対戦ゲームではこれで「場」はさびれてしまうかもしれません。「場」がさびれるともうゲームの盛り上がりは期待できません。ゲームはつまらなくなります。すると，もうそれはネットを伝わって「あのゲームはつまらないらしい」という評判になります。そのせいでゲームはもっとつまらなくなります。ああ，悪循環。

スタンドアロンのゲームではルールをまず覚えることからゲームの楽しみにすることもできますが，インターネット対戦ゲームでは，ルールはそのゲームをプレイした誰もがわからなければならない程度には単純であるべき，そしてみんなの暗黙の了解から外れるべきではないのです。まぁ，ネット麻雀をやる人は，ある程度麻雀のルールはわかっているもの，とできるでしょうが（というか麻雀は食いタンあり/なしくらいしかルール変えられませんものね），要は「場」として盛り上がるタイプのゲームでは場にいるみんながわからなくちゃいけないよねぇ，というわけです。

世間一般のコミュニケーションにも当てはまる常識だとは思いますが，たとえば，ひとりで飲みに行くときは飲みながらシェークスピアだろうがオライリーの動物が表紙の本（わかりにくいたとえですが，要は世間一般の人にはあまり理解できない難しい本，という意味です）を読んで楽しんでも構いませんが，多人数で飲むときにはなるべくみんながわかるような話題でも話してみんな盛り上がれるようにしたほうがいいでしょ？　そういうことです。

＊

さて，ここまで書いたことはインターネットゲームを含めて（特に最近の）ゲームというものが「場」である，ということさえ意識していれば，まぁ，踏み外しようがない鉄則のような気もします。でも，特にパソコン用のインターネットゲームとして最近発表されているもの（特に国内モノ）にはなにかこれを忘れているような気がします。

本当にいまのゲームに必要なのは優れたモノ書きでも音楽家でもありません。いや，いてくれることにこしたことはないんですけどね。でも，あえて，いわせていただきましょう。いま，いちばんPCゲーム業界に欠けているのは，「宴会の幹事」なのであります。わーっと人を集めて，ぱーっと盛り上げてくれる幹事。この才能こそがやっぱりいちばん必要でありましょう。

ささ，そういうわけですから，これを読んだゲーム業界の皆さま，私をぱーっと飲みに連れていってやってください。飲ミニケーションというやつです（わぁ，このオヤジくさいいい方）。そうやって世間の暗黙を探るのです。人の意見というのは貴重ですよ。お誘いに遠慮はいりません，もちろんあなたの会社の経費で落としていただいて結構ですから（笑）。

図4　この辺を見ると普通の信長っぽい

図5　秋くらいには攻め込まないと……

図6　メニューなどはこんな感じ。とまどう人も多い

SPECIAL

特集 ネットワークゲームの地平へ

海外コインオペゲームにおける通信

市川幹人 *Ichikawa Mikito*

コインオペの世界では格闘ゲームの筐体を２台つないだり，レースゲームを複数つなげて多数のプレイヤーを競わせるゲームは数多く存在しています。機器のつなげ方だけを考えてもさまざまなアイデアがあるはずです。ここでは海外のコインオペゲームを２点紹介します。

■Golden Cue

Sega Pinball社とIncredible Technology（以下，IT）社の共同作品。世界中のGolden Cueとセンターを通信で結び期間単位での最高得点を競わせ，入賞者は賞金を受け取れるという狙いの作品。日本でも格闘ゲームを中心としてゲーム大会が行われるが，これを世界規模でしかも賞金をかけて行おうという試みだ。自分のレベルを知ることができると，多くのプレイヤーにとって励みになることが多く，そのジャンルのプレイヤーの増加にも大きくつながる。

IT社からはWilliams社で数々の作品を残したMark Ritchie氏（Indiana Jones，Fish Tales，Taxiなどは日本のファンにも馴染みがあると思う），Sega PiballからはGottlieb社最後の10年を支えたJon Norris氏がこのプロジェクトをリードしている。Sega Pinballはピンボールのノウハウを，IT社は通信ゲームのノウハウを持ち込んだ。

IT社は日本ではほとんど馴染みがない人が多いが，現在米国でゴルフをテーマにしたGolden Tee 3Dがロングヒット中，この作品でIT社はコインオペ業界での通信技術を確立している。また，ITはピンボールでは，Sega Pinballを前身であるData East Pinball社の初期の作品でプログラム部分を担当しており，ピンボールとも無縁な会社というわけでもなかったようだ。

■トーナメントであるための工夫

いままでのピンボールではプレイ時間が長い＝高得点だったが，Golden Cueでは決められた種目をいかに短時間にクリアするかが重要だ。すべての種目はゲーム開始からクリア得点がカウントダウンされる，したがって，長く遊ぶことは高得点につながらない。すべての種目を終了した場合もゲーム終了となる。

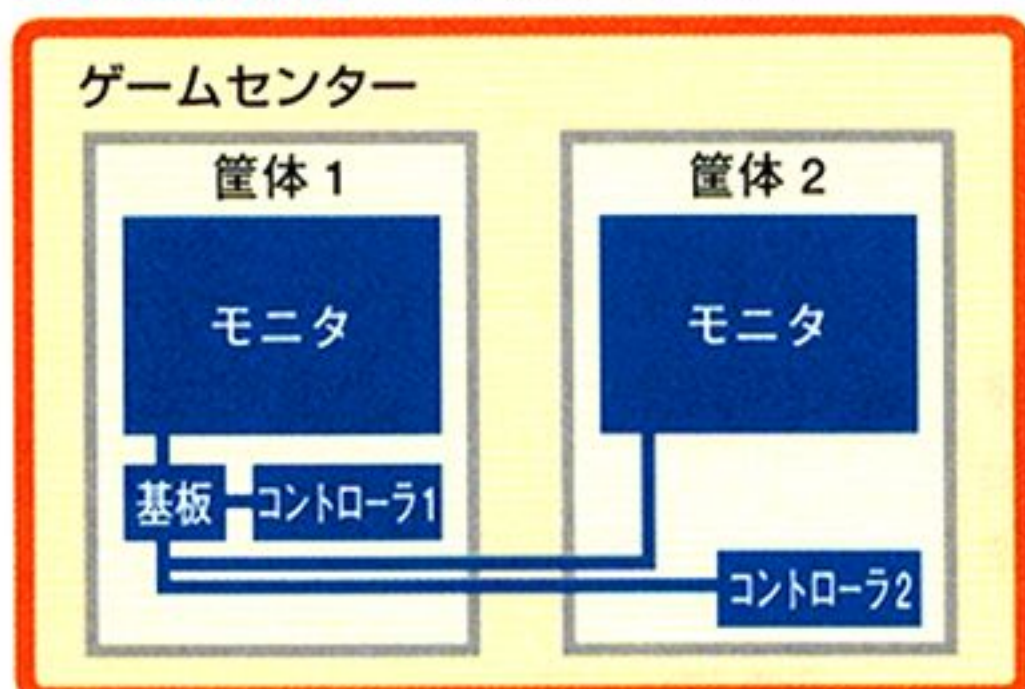
なぜこのような形態になったのか？　自然発生して進化した遊びというものは，多くの年月を費やして現在の形になっている。この過程で，あまりにも安易な必勝法のある遊びは改善されるか，または淘汰されてきている。しかし，コインオペのゲームは自然発生した遊びに比べ，大変短い期間で完成される。実際ビデオゲームでは多くのゲームに永久パターンが存在している。賞金のかかった期間中にルールのバグなどが発生した場合「長く遊ぶ＝高得点」の形式を採用すると致命的な欠点になるのでカウントダウン制が採用されたと筆者は考える。

これから通信ゲームを考える人で，特に相手プレイヤーの操作するキャラクターに直接攻撃可能なゲームを作る場合は，いわゆるハマリを防ぐことと，Golden Cueのように構造的にハマらなくなる方法を考えるべきだろう。また，考える段階で自然発生した遊びのルールや歴史を研究するの

写真１　ショウに出展されたGolden Cueデモ機。ドロップターゲットに絵が書き込まれてない辺りが未完成であることを感じさせる

写真２　左下にある緑のチェックマークに注目。クリアした種目はライトが点灯し終了したことを示す

写真３　バックグラスに上にある電光掲示板が目を引く！ここにいろいろな情報が表示される。筆者がプレイしたときにはまだ完成してなかったため，たいした情報は表示されなかった。完成版では"リアルタイムのスコア更新情報""新作のアピール""イベント情報"などが表示されることが容易に想像できる

●対戦格闘ゲーム（例外もある）
ゲームセンター
筐体1
筐体2
モニタ
モニタ
基板
コントローラ1
コントローラ2

●Golden Cue　センターと結ぶところが新しい
Shopping mall
Golden Cue
センター
遊園地
Golden Cue
ゲームセンター
Golden Cue
バー
Golden Cue

●Scorpion DX（S.DX）
マシンとパソコン，インターネットまで活用。
パソコン上ではLeague Trackerという専用ソフトで管理，編集が可能
S.DX
S.DX
S.DX
DOS/V
DOS/V
DOS/V
プリンタ
インターネット
プリンタ
インターネット
プリンタ
インターネット
トーナメントHP
メーカーHP等

図1　コインオペゲームの通信形態

も有効だ。

■センターと通信するメリット

ピンボールは実物の鉄球がフィールド内を動くため，どうしても故障が発生する。現在，Williamsのピンボールでは10ゲーム以内に一度も反応しないスイッチがあると，クレジットの横に“.”を表示して故障しているスイッチが存在することをアピールする。さらに，そのスイッチが存在しなくてもゲームが問題なく動作するように構成されている。これはメンテナンスを行う意識のある店舗ではパーツ発注などの時間まで「とりあえず動作してくれる」ため，大きなメリットとなるが，半面，それ以上にメンテナンスする意識のない店舗を多く生み出してしまった。ピンボール台には日々のメンテナンスが不可欠だ。壊れても台が知らせてくれるという油断が，日々のメンテナンスをおろそかにさせる。せっかく警告が出ていても本当に壊れる前に部品交換をしなければなんの意味もないのだ。

これがGolden Cueではどうだろう？　通信回線で連絡できるのだ。自動的にセンターに故障情報を転送することで，店舗に対する警告を発したり，サービスマンを迅速にメンテナンスに向かわせることができる。また，賞金をかけて競わせる作品なだけに，台ごとに条件が違うのでは大変な不公平であり，致命的な欠点になりかねない。それを防ぐ意味でも大きなメリットだ。

近い将来，通信回線がさらに整備されれば，あらゆるコインオペ機器に通信機能が搭載されるだろう。プレイヤーは故障した機械や調子の悪い機械で遊ばされることはなくなり，結果的に店舗の売り上げも増加することだろう。

家庭用ゲーム機やパソコンの進化が激しく，コインオペテレビゲームの将来が危惧される現在，遊戯場では家庭でプレイできない，機械の絡んだゲームが再び注目されるだろう。機械の絡むゲーム機には，黎明期から存在する最大の問題だった「故障」を激減する可能性があり，このアイデアはコインオペ産業に大きな影響を与えそうだ。

■Scorpion DX

近年注目を浴びている，コンピュータでスコアなどの管理を行うコインオペ方式のダーツだ。ピンボールやテレビゲームが，誕生から間もない段階でコインオペになったのとは対照的に，大変歴史のある遊びが現代でコインオペになったことはそれだけでも注目に値する。

コンピュータが搭載されたゲーム機はゲームの進行役のみを担当する。ほかのコインオペゲームではゲーム機はゲームの進行役そしてプレイヤーの敵の役割を担当していた。フリッパーの登場以降，長らく続いていた機械vs人間という図式ではないコインオペゲームだ。

通信を行い，リアルタイムで他店舗のプレイヤーとプレイ可能なだけではなく，パソコン上の専用ソフトウェアでプレイの傾向の解析，過去の戦績の管理が可能。このデータをプリントアウトしたり，インターネット上で交換，公開することで競技人口，プレイ人口の増加を狙っている。

クラシックな遊びはプレイが形になるまでに多くの時間を必要とする。テレビゲームは誕生から間もなく産業になったため，各社が鎬を削った結果，「すぐにプレイが形になる」（上級者になった気分になる），「初心者に異様なまでに親切」な作品が多くなった。「好きこそものの上手なれ」というように，人はできることには興味を示しやすい。この結果テレビゲーム登場以降，クラシックな遊びは大幅にプレイ人口を失っていった。

しかし，コンピュータ管理を導入し，さらに通信を導入することでクラシックな遊びの世界にコンピュータの利点を活かし，生き残りの道が開けてきた。もともとゲーム性の奥の深さは抜群なので，商品としても成功すると思われる。実際，コンピュータ化されたことで，これまでダーツなど敬遠していた若いユーザー層の注目を集めて，非常に幅広い層の支持を獲得している。

競技レベルのものを除き，クラシックな遊びでは，ほとんどのプレイヤー（競技レベルのプレイヤーはアスリートと表現したほうがよいかもしれない）はプレイすること自体と同時に，ほかのプレイヤーとの会話なども楽しんでいる。通信技術の進歩は物理的に遠く離れたプレイヤー同士での対戦プレイを可能にし，実際に移動することなく，いつでも他人とプレイを可能にしてくれた。これはこれで素晴らしいことだ。しかし通信媒体に依存しない「自然発生した遊び」の面白さももっと再認識されることを筆者は望みたい。

Scorpion DXのホームページ

Web上ではハイスコアランクに入ったチームの写真が公開されるなどの盛り上がりを見せている。ユーザー層の幅も広そうだ

写真4　Scorpion DX　モニタもついて，クラシックな遊びをわかりやすくしようとしている。テレビゲーム世代のプレイヤーにも遊びが広まるようにしている

アーケードに見る通信対戦ゲームの未来

如月 緑 *Kisaragi Midori*

「通信ゲームなんて回線が遅くて，全然現実的じゃない」と嘆く人もいるでしょうが，だからといって手を出さずにいると時代に取り残されてしまいます。高速回線でネットワークゲームができたらどうなるのか？　そのひとつの答えがアーケードゲームなのかもしれません。

ネットワークゲームとは，いくつかのコンピュータをなんらかの通信手段で繋ぐことを前提にして作られたゲームのことである。

一般に，複数人で同時にプレイできるものをリアルタイムネットワークゲームと呼び，その他，コンピュータが通信できることを利用しているだけのもの（得点を集計するなど）とは，区別するようだ。昨今では，パーソナルコンピュータによるインターネットを使った通信対戦がごく当たり前に行われるようになってきた。

一方で，遠隔地対戦という部分を除けば，業務用ゲーム機はかなり以前からリアルタイムに通信対戦をすることが可能だった。家庭用ゲーム機はコスト的な制限から，なかなか通信ゲームは作られずにいるが，徐々にインフラも整備されてきて，これから普及する機会をうかがっているかのようである。

今回はこうした状況のなか，ネットワークゲームに必要なプログラム構造などについて考えてみよう。

■ネットワーク構造とゲームの可能性

ネットワークを構成するために最低限必要なのは，2台以上のコンピュータである。

当たり前だが，なんらかのケーブルや赤外線，電波などで実際に繋がっていないとネットワークを構成することができない。ケーブルをどのように繋いでネットワークを構成するかと，実際にそのケーブル内をどのような信号で通信するかという部分で，それこそ何冊も本が書けてしまうほど数多くの方法がある。だが，どういった通信手段を用いたとしても，リアルタイムで通信するゲームを構成するうえで問題となるのは，以下の部分である。

- ・ゲームサーバがあるかないか
- ・最大何台で通信するか
- ・通信レート（速度）が高いか低いか
- ・通信遅延時間が大きいか小さいか
- ・安定しているか

まず，ゲームサーバがあるのかどうかは，その上で実現するゲームの種類や方法に大きく関わる部分である。ゲームサーバの作業は，基本的には同時プレイの仲介とデータの受け渡しなどであるが，これが専門に存在するかどうか，あるいはこの部分がどれだけ高性能であるかで，繋がったクライアントのプログラムの構成は大きく異なる。

また，最大何台で通信するかということも，単純に通信量が増えるだけではなく，それぞれの処理量に大きく関わる部分であろう。

通信遅延時間が安定しているかどうかは，ゲームの通信プログラムの構造に，大きく影響を与える部分である。これが確実に安定していると，プログラムの構造は非常に単純化できる。

■業務用ゲームでの通信環境

業務用のゲーム機では，遥か昔からリアルタイム通信ゲームが存在していた。たとえば大型筐体もののドライブゲームなどである。しかし，これらは当然一部の例外を除いて，どこか遠隔地のコンピュータと通信しているわけではなく，隣あっ

図1　アーケードゲームではサーバを置かず，対等に並んだゲーム機のどれか1台が仮にサーバ役を行う

たゲーム機同士が互いに通信しているにすぎない。

また，最近になって若干の本格的な通信ゲームが出始めてはいる。だが，ゲームのリアルタイム性を失わずに通信することを考える場合，遠隔地との対戦は，まだ乗り越えなければならないハードルが高い。

さて，現在実現されているドライブゲームなどの業務用の通信ゲームでは，その形態から，以下に挙げるようにいくつかの特徴がある。

- ゲームのサーバはないことが多い
- 通信遅延時間が小さい
- 通信レートは高い

業務用のゲームの場合，ゲームのサーバは通常は存在しない。なぜかというと，業務用ゲーム機を運用するとき，台数は不定なのだ。つまり，1台だけで遊べるように設計してあるものを何台か導入し，そのまま通信できることが必要なのである。どの機械も同じ構成になっていないと効率が悪いのは簡単に理解できるだろう。ただし，ゲームの進行を司る部分は必要になるので，同時にゲームスタートした台の中で，どれか1台はそういった管理役になる必要がある。

また，通信遅延時間が許容されるかどうかは，ゲームの内容によるところが大きい。これまでは業務用の通信対戦といったら，一般にアクション性の高いゲームで行われてきた経緯があり，通信の遅延や不安定さはゲーム性に著しく影響を与えるので，できるだけ短いことが要求されてきた。

最後の業務用ゲームの通信レートが高いのは，単純に遠隔通信をしないからにほかならない。なにしろほかの種類のゲーム（＝コンピュータ）と繋ぐ必要もなかったので，通信自体は独自規格でよく，コスト面でも制約は少ない。現在では光通信を使用して非常に高速な通信が可能となっている。

業務用のゲームでは，以上のように通信環境としては非常に独特であるといえるだろう。通信ゲーム環境としては理想に近い要素が多く含まれている。現在の一般的な「通信ゲーム」環境と比べて恵まれているが，将来的にはこれくらいの環境が一般家庭を結ぶ回線でも期待できるのだろうか。ある意味で現在の業務機の通信ゲームは（少し遠そうだが）将来的な広域ネットワークゲームの直面する課題を先取りしているともいえるだろう。その通信速度，ハードウェア環境……，すでに未来の姿は提示されているかもしれないのだ。

■家庭用ゲームでの通信環境

家庭用ゲーム機では，これまで通信環境と呼べるものはほとんどなかった。

しかし，最近のゲーム機では一応通信ができる状況にあり，たとえばセガサターンでは，X-BANDなどで通信ゲームができる。だが，基本的にこれらの通信はリアルタイム性には乏しく，データの配信や集計にのみ使われることが多かった。しかし，インターネットを介しての通信環境が徐々に整備されてくるにつれ，パーソナルコンピュータの世界では，リアルタイム通信ゲームがどんどん出現してきている。したがって，その遊ばれ方が類似している家庭用ゲーム機で，遠からず同様なゲームが出現するであろうことは想像に難くない。

ただし，家庭用ゲーム機では，その購買層からいって通信コストが非常に問題になるだろう。家庭用ゲーム機での通信環境は，速度や通信方法に関していえば，インターネット接続環境そのものである。したがって，以下のような特徴が挙げられるだろう。

- ゲームサーバがあることが多い
- 通信遅延時間および，通信レートは非常に低く，かつ不安定である

これらの特徴は，家庭用ゲーム機だからということではなく，どちらかというと，接続にインターネットを用いる形態だからである。

ゲームサーバは，不特定多数の同時プレイをサポートするために，専用で存在しなければならない。通信遅延時間も，インターネット経由になった時点で，ほとんどあてにならなくなる。これは，インターネットそのものが，低品質の通信について特に許容するシステムなので，やむをえない部分である。

■通信ゲームで必要なアルゴリズム

まず，一般的に用いられるゲームの構造について，簡単に説明しておこうと思う。

通常，業務用，家庭用を問わずゲーム専用機では画面のリフレッシュされる時間を処理の最小単位とすることが多い。これはもともとテレビやコンピュータにおける映像出力が，人間の目の残像を利用して，単位時間に1枚ずつ画面を生成するシステムだからである。つまり，その単位時間内は出力されている映像が変化することはない。それゆえ，画面の表示に必要なありとあらゆる計算はその時間を単位として回すのが最適である。

また，昨今のゲームでは，ポリゴンなど，3次元CGを生成する構造が多い。通常，3次元CGをリアルタイム生成する場合，ジオメトリ計算およびレンダリング計算に1単位時間をまるまる使用する構造にして，表示はその次の1単位時間で行われることが多い。入力デバイスなどはこの単位時間に1回ずつ，読まれることになるのである。

■通信ゲームでの通信手法

通信ゲームで送るデータについては，どういった通信手法をとれるかによって，大きく異なる性質がある。

まず，考えられるのは，お互いの空間上の位置や向きといった情報と，お互いが持っている状態（体力や装備品などのパラメータ），それらがデータとして意味をなすようにするための時刻を送りあうことである。

この方法は，データとしては申し分ない情報量を持てるのだが，大きな問題点がある。それは，データの量が多すぎることである。

遙か昔のゲームならともかく，いまどきの3次元CGゲームでは座標や向きを正確に画面に反映できるので，空間上の位置などをちゃんと送らないと，互いに相手の画面上では，異なった位置に物体が表示されてしまうのだ。

そのため，位置および向きの情報だけでも，6個のデータを送らなくてはならない。

たとえば，理想的な遅延条件で考えることにするとして，

条件1：あるデータをゲーム機から送信し始めた時刻から数えて，そのデータを送り先のゲーム機が受け取り始めるまでの

図2　座標データなどを送るとCPU負荷は軽くなるのだが，通信回線の負荷は非常に大きくなってしまう

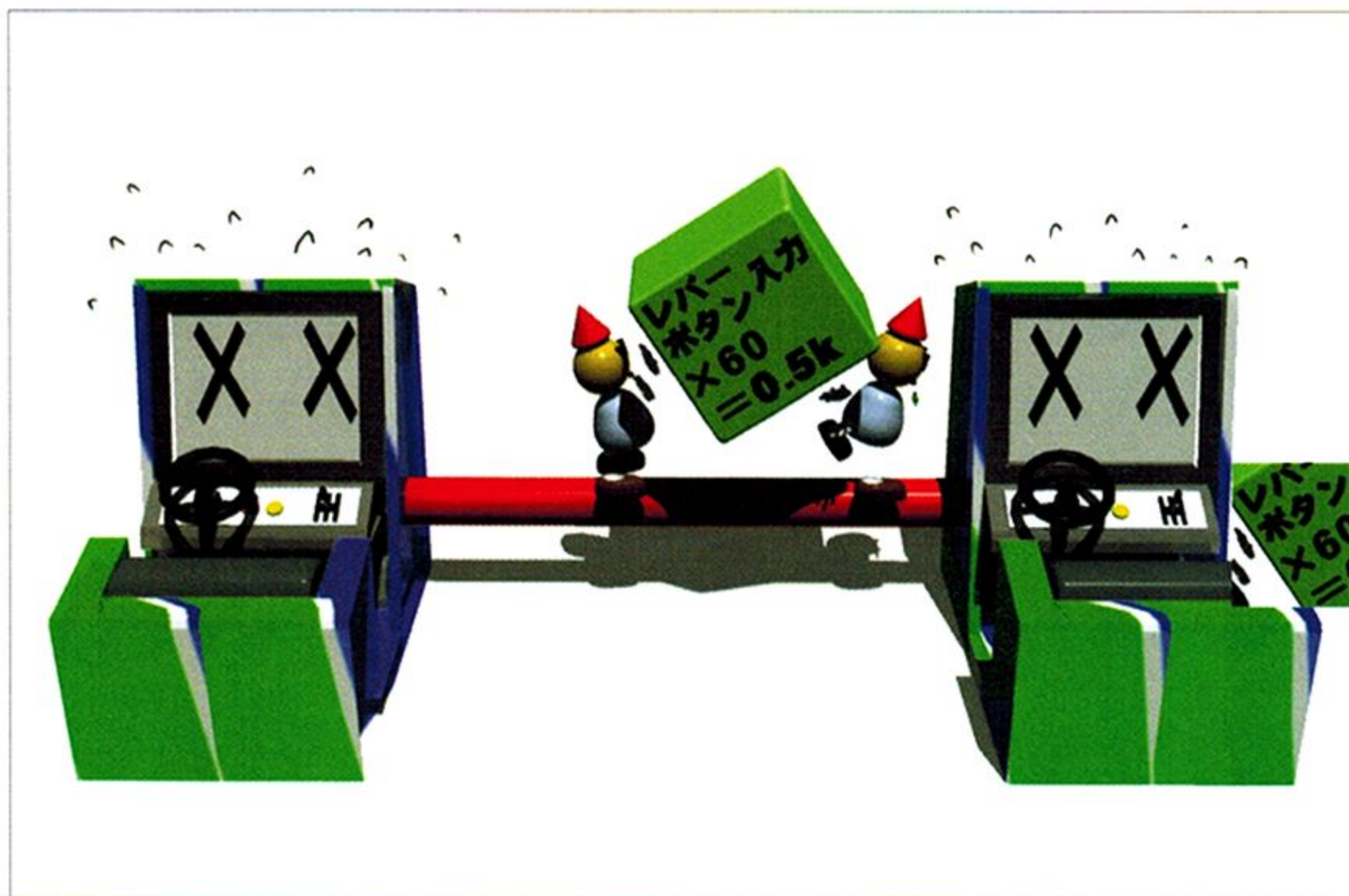



図3　入力データを相互に送りあうと，回線は非常に軽くなるのだが，それぞれのCPU側に負担がかかる

時間（通信遅延）はほぼ無視できる
条件2：ゲーム機同士は直接繋がっており，接続台数は2台である

この条件で，先ほどの位置および向きを互いに送りあうものとする。

なお，通信の遅延は条件1により存在しないとして，データの時刻は送らないことにする。

データはすべて単精度のIEEE浮動小数点データであるとして，

1台1回分のデータの大きさ
＝4バイト×6（個）＝24バイト

1回の計算は普通画面書き換えの時間と同期させるので，通常1/60秒で計算するとすると，1秒間に送らなければいけないデータの大きさは，

60（回）×24バイト＝1440バイト

通信の世界では，データの大きさは普通ビット単位で，スタート/ストップビットなどを考慮した一般的な通信フォーマットの場合1バイトのデータは10ビットで送受信されることになる。全二重モデムを使う場合，送受信は独立して行われるので，まとめると，

1440×10＝14400ビット

となる。

つまり，たったこれだけの条件でも，実際に通信するためには14.4kbps以上のスピードが出せるモデムが必要になるのだ。確認しておくが，ここで想定されているのは，2人のプレイヤーごとに1個の物体の3次元座標と向きのデータのみを送ることである。現実のゲームでは，もっと多くの情報を，もっと多くの台数でやり取りするので，この方法ではデータの大きさが非現実的になることは簡単に予想できる。

ほかによい方法はないのだろうか。手法を変えて，データを計算する前の，もっとプリミティブな情報だけをやり取りすることを考える。

まず，前述の条件でできる2人同時プレイのゲームがあるとする。

このゲームのプログラムは，2人分の計算を同時にまったく同じ条件で行うとして，片方の計算に使う入力信号はそのゲーム機に直接繋がったボタンやレバーなどから，もう片方の計算に使う入力信号は，そのゲーム機に送られてきたデータを使うとしたら，どうだろうか。

この場合には，先ほどのデータ計算でいうところの，1台1回分に相当するデータは非常に小さくて済む。入力データが，たとえばアミューズメントパークに置かれている8方向レバー＋6ボタンのデジタル入力であるとすると，8方向レバーの入力は3ビットの情報量，ボタンは1個1ビットの情報量しかない。先ほどの例に当てはめると，

1台1回分のデータの大きさ
＝3ビット（レバー）＋6ビット（ボタン）
＝9ビット

したがって，1秒間に送るデータの大きさは，

60（回）×9ビット＝540ビット

計算の都合上，初めからビット単位で表記した。この条件なら，1kbps以上のスピードがあればいいので，結構現実的なのではないだろうか。

現在リアルタイム通信ゲームでインターネットを介するアクション性の高いものは，こちらの手法をとるのが普通であろう。

ただ，この方法に問題点がないわけではない。実際に使うには最低でも適切なエラー訂正符号化をしないと，相手が送ったつもりのデータとこちらが受け取ったデータが違っても対処できない。この方式の場合は特にデータエラーが致命的となるのだ。

ほかにも，通信データとしてお互いの入力デバイスのデータのみをやり取りする場合には，ゲームを作るうえで厄介な問題が潜んでいる。

まず，ゲームのプログラムをお互いのプレイヤーに対し，非常にシンメトリカルに作らないといけなくなる。

さらにこの方法では，同時対戦プレイヤーの数に比例する処理量をゲーム機がこなさなくてはならなくなる。最大同時プレイ人数を大きくすればするほど，ひとつあたりに使える処理量は小さくなるのだ。

接続台数の変化によって挙動プログラム（処理量）の変わるようなゲームは，繋がっている台数によってプレイした印象が変化するので，よい方法とはいえない。だからといって，最初から最大接続台数を見越した処理量に抑えることは，そのゲーム機の性能をうまく引き出したかどうかといった点で問題があるのだ。

ちなみに座標を送りあう場合，この処理量は，ほとんど変化しない。なぜなら，通信相手の計算をしなくてもよいからだ。

また，プレイヤーが互いに干渉しあう場合，つまりお互いのコリジョン（接触判定）を考えると少し面倒なことになる。コリジョンは，お互いのデータを，過不足なく同じ手法かつ，同じ順番で処理する必要がある。プレイヤーになんらかの影響を与える移動物体などは，すべて繋がったゲーム機で同じ処理をしなければならない。たとえば，単純に処理するとAとBの衝突判定などで，Aの処理をしてからBの処理をするのと，Bの処理をしてからAの処理をするのでは結果が変わってしまう。ちゃんとしないと，昔あった対戦ゲームみたいに，同じ条件のときはAが必ず有利になる……などという不公平も発生しかねない。

また，ある瞬間にプレイヤーに関係ない部分であっても，どこかで不一致が発生すれば，いずれプレイヤーにわかる部分で異なる結果が発生してしまう。

以上の問題により，業務用ゲームでは専用の高速光通信ができることを利用して，座標そのものを送ることも少なくない。通常，この問題は，通信の品質がどの程度まで保証できるか，あるいは，通信自体にかかる処理量などによって，切り分けて考えるのが望ましい。

■通信ゲームで送るデータによるアルゴリズムの選択

通信ゲームでは，前述の転送データの違いによって，ゲーム自体の考え方が変わってくる。

最初の，座標などのいわゆる計算結果のデータをやり取りする手法で通信する場合には，通信ゲーム特有のアルゴリズムや考え方はさほど多くない。

本来通信しないゲームのアルゴリズムで考え，自分とコンピュータ操作の敵がいる前提でプログ

ラムを組んで，たとえば敵のうちのどれかに，通信でもらってきた座標などを上書きするかたちで作れるのである。

この場合，注意するのは通信してきたデータに，どのくらいの時間遅れが生じているかという部分だけであり，その時間が数回計算分程度以内に収まる場合には，速度や加速度，また角速度や角加速度も同時に送って，その計算回数分ただ単に足し込んで（現在位置の予想を）計算して誤魔化すのである。この方法ならば最高のレスポンスが通信時でも失われない（衝突判定などのイベント発生は実際のデータがくるまで保留すべきだが）。

ここで生じた本当の位置からの誤差は，毎回座標データを送れるのであれば，毎回修正されるので，最大でも足し込んだ回数分の誤差しか発生しない（誤差は累積しない）。

そして，この数回分という，ゲーム性が破綻をきたさない範囲でデータを通信できない場合には，この方法は向いていない。なぜなら，現在位置の予想そのものに高度な計算を要求するのは本末転倒だからである。

次に，入力デバイスのデータなどを送る手法をとった場合，もっと違った点で工夫しなければならない。まず，通信する台数を想定し，その台数分，プレイヤーのプログラムをループして計算することは当然必要である。

また，まったく別々のマシンで同じ計算結果を得るために，コリジョンや，プレイヤー相互のデータを参照するようなプログラムでは，全体の計算を一度に行うことは大変なので，1台ずつの計算を順番にやって間にあわせることが多い。当然ながら，この場合の計算の順番は，システム全体で決めた手順で行わなくてはならない。

例として，8人同時プレイ可能なゲームで，5人が同時にゲームをしているとする。5人は，それぞれ，1，2，4，6，8番目のゲーム機でゲームをしているとすると，それぞれのゲーム機の計算に求められる順番は，

1台目：自→2 →敵3→4 →敵5→6 →敵7→8
2台目：1 →自→敵3→4 →敵5→6 →敵7→8
3台目：1 →2 →敵3→4 →敵5→6 →敵7→8
4台目：1 →2 →敵3→自→敵5→6 →敵7→8
5台目：1 →2 →敵3→4 →敵5→6 →敵7→8
6台目：1 →2 →敵3→4 →敵5→自→敵7→8
7台目：1 →2 →敵3→4 →敵5→6 →敵7→8
8台目：1 →2 →敵3→4 →敵5→6 →敵7→自

ちなみに，ゲームに参加していない3，5，7番目のゲーム機は，ライブ中継をしているものとし，その分のプレイヤーはコンピュータ操作とする。これを，不公平のないように，順番を入れ替えながら行わなくてはならない。優先順位をローテートしてやればマクロ的視野で見て公平な結果が期待できる。

また，プレイヤーになんらかの変化を与える移動物体の接触判定や，動きの計算も，同様にして順番を規定して計算する必要がある。この方法では，自分の画面内で表示されているかどうかに関わらず，ありとあらゆる計算をすべて同一条件で行う必要があるのだ。

また，先ほどと同じように，通信してきた入力データに，どのくらいの時間遅れが発生しているかによって，ある程度細工をする必要が生じる。

この場合の入力データの遅れについては，自分の入力データを，擬似的に遅れ分だけバッファリングすることによって，ゲームのレスポンスの遅さと引き替えに，ある程度は補償してやることが可能である。

たとえば，通信にかかる時間が計算回数にして2回分で，必ず一定という場合には，自分の入力データもすぐには使わず，2回分だけ記憶しておき，2回分古いデータを使うことによって，入力データの時間差を相対的になくしてしまうのだ。

この場合，当然ながら，たとえひとりでゲームをしたとしても，自分が入力したデータが画面に反映されるまでに2回分の計算時間が余計にかかることになる。

これがゲーム性に問題になるくらい大きいようであれば，やはり予測動作が必要になる。入力デバイスがアナログ信号からのA/D変換値であれば，突然大きな変化量を持つことはあまりないので，しばらくの間は，人工知能プログラムにでも予測させれば問題はない。ただ，この場合には，予測と実際との差異によって，計算結果が変わってしまう可能性が高いので，正しい入力データが通信されてきた場合には，そのデータ位置まで遡って計算しなおすか，あるいはときどき定期的に計算結果も通信して，データを修正する必要がある。

大味な海外産のPC用ゲームでは入力遅延などはほぼ無視しているようだが，日本のお家芸といえるアーケードクラスの対戦格闘ゲームやレースゲームでは，わずかなレスポンス遅れでもゲーム性に大きく影響が出てくる。実際にアーケードゲームではこういった予測動作による入力遅延の解消が行われており，高いゲーム性が保たれている。

■通信ゲームにおけるゲーム進行の管理

通信ゲームを構成する場合，誰がゲームの進行役になるかという問題がある。ゲームを管理するサーバがいれば，そのプログラムが，残り時間や参加プレイヤーの状態に応じて，ゲームの進行をすることになる。また，そういったサーバを介さないでゲーム進行をする場合には，同時にプレイしているゲーム機のうちのどれか1台が，その役割を行っている場合が多い。

まず，管理用のサーバがある場合には，ゲームルールの進行はあまり問題にはならない。ゲームの結果もサーバが決定したものをすべてのゲームクライアントが受信しさえすれば，問題は特には生じないはずである。

ただし，前述の予測動作をしている場合，あるゲーム機が先に判断した結果と，公式結果が異なることもありうるという前提で，プログラムを組んでおかなければならない。

図4　予測動作には実際の確定座標との誤差がつきまとう

通信管理以外をサーバが行っていないとした場合，なんらかの形でゲームの進行に必要なデータを，サーバが常に受け取る必要がある(たとえば，ゴールしたかとか死んだかなど)。

また，管理用のサーバがない場合には，ゲームルールの進行はゲームの参加台のうち，どれか1台がその役目を果たすことになる。

ここでゲームの参加台のうち1台と決めたのにはそれなりにわけがあって，このゲームの進行というのは処理量は重くないものの，必ず定期的に行われる必要があるので，そのゲーム通信のデータを直に拾える，つまりそのゲームに関わっているゲーム機が行ったほうが都合がよいのだ。

通常，この動作形態は，業務用の通信ゲームで多い。たとえば業務用ドライブゲームの通信では，誰か参加する人のエントリーを待つゲームの性格上，必然的に，残り時間などを管理するのは，最初にお金を投入して選択画面に移ったゲーム機で，制限時間内にエントリーした台とのゲームの進行を管理することになる。

それから，サーバがあるないに関わらず，なんらかの決定をした場合の通信遅延による構造上の問題点を考慮する必要がある。

まず，残り時間でゲームの対戦相手を待っているようにできている場合，その時間切れをサーバ側が決定して，それを通信してゲーム機が拾うまでに，ゲームにエントリーした人を入れるかどうかという問題がある。

入れるのであれば，再調停して参加台数を増やし，また通信しなければならない。

入れないのであれば，ゲームにエントリーした人について，納得のいく画面を出さなければならない(ゲームにエントリー直後，サーバからエントリー確認の通信が来るまでの間，画面上にエントリーを受け入れたようにとられる映像を流してはいけないなど)。

また，予測動作をしている場合，たとえばゲーム終了条件を満たしているにも関わらず，サーバから終了の信号がこない間はゲームが続行してしまうと，非常に不可解な画面を表示してしまうことがある。

この場合を考えると，その台で独自にゲームを終了させて，結果の調停だけをいくらか待つ構成にしたほうがよいことが多い。

そのため，ゲームルールの進行は，いずれ全体として起こることの確証が取れている場合には，それぞれのゲーム機が独自に進行させて，順位などはあとから調停するようにするとよい。

通信ゲームの場合はどんな構成であれ，フェイルセーフを心がけなければならない。

さらに通信ゲームを制作するとなると，ほかにいくつか考えなければならないことがある。たとえば，通信対戦によってゲーム仕様の論理的な不具合が出てくる可能性が考えられる。このことは，たいていゲームの制作上，仕様策定のタイミングではすでに考慮されなければならないのだが意外に忘れられがちである。

まず，残り時間でゲームの対戦相手を待つとき，エントリーを受け付けると，残りの時間をいくらか増やす仕様になっているとする。

このこと自体は一見サービスしているように見えるので，よさそうなのだが，最悪の場合，最初にエントリーを受け付け始めた時点のイニシャルの残り時間と，1エントリーで増やす時間×エントリー可能最大数の総和時間，最初にエントリーした人が待たされる可能性があることを十分に吟味する必要がある。

たとえば，8人同時対戦可能なゲームを，イニシャルのエントリー待ち時間が10秒，1人エントリーするたびに5秒サービスすると，最悪で10＋5×7＝45秒は待たされる可能性があるわけだ。

この時間は，家庭用ゲーム機やパーソナルコンピュータで行うゲームとしては，まあ我慢できるかもしれないが，業務用ではアウトである。

また，通信が途絶えた場合のことを考えておかなくてはならない。

どのような条件で，どちらから通信途絶を判断するのか，そして，切り離したプレイヤーを，どのように扱うかである。

ゲーム機の場合，それほど大きな問題にはならないので，通常は見すごされがちだが，本来，リアルタイムで処理を行う場合には，ちゃんとしたフェイルセーフな構造が必要である。この部分に関しては，通常，リアルタイムシステムとしての通信処理そのものの構造のよし悪しが，性能を決めることが多い。

■通信ゲームの将来

現状のアーケードゲームというかなり特殊な通信環境を中心に述べてきたわけだが，確かに一般的なパソコンやゲーム機とは事情が異なる面も多い。しかし，参考になる点も少なくないだろう。

これからの通信ゲームを創造するうえで，避けて通れないのが，インターネット通信のコストである。一般家庭をターゲットにするとインターネットの通信網をあてにせざるをえないのだ。実際，欧米に比べ，特に日本の通信コストは極端に高いことがいわれている。しかし，そのコストは下がってきており，将来的には，十分に高速な回線が日常的に使用可能になるであろう。

そういった通信速度が十分に速いということを前提にすると，サーバにすべての動作をさせるシステムの構築も考えられる。

かつてナムコが行ったドライブゲームのテストでの例であるが，通信回線にはATMを使用し，東京と大阪の2拠点からの通信データを横浜の未来研究所に集め，遠隔地対戦の実験を行った。このときの通信は，入力データを東京および大阪から通信し，未来研究所内に隣同士で通信しあっているゲーム基板に入力し，映像と音はリアルタイムMPEG圧縮をかけて送信したものである。

このシステムで重要なのは，東京と大阪の2拠点に置かれていたゲーム機には，ハンドル，アクセルなどの入力デバイスと，モニタやスピーカなどの出力デバイス以外には，通信に必要な機材しか入っていないことである。

つまり実際のゲームとしての通信は，研究所内で，わずか1メートル以内の範囲でしか行っていないのである。

これは，通信速度が非常に速くなければ成立しえない手法である。たとえばこれが普及すると，すべてのゲーム機は，ちょうど昨今いわれているネットワークコンピュータのように，必要最低限の入出力デバイスと通信機器のみで構成され，ゲームそのものはサーバが行うことになるかもしれないのだ。

どんなに家庭用ゲーム機が進化しようと，その容量や性能には限界がある。ネットワークを前提とした超巨大ゲームなどが登場すると，ゲームに関する大半の処理は超並列のサーバコンピュータなどが担当することになるのだろう。画像生成もサーバ側で処理してMPEGで配信……という先ほどの例は極端すぎて現状の通信回線を考えると鼻白む部分もあるのだが，それもすでに実際に行われたことなのだ。

現在最新のMPEG4は一般通信回線でのリアルタイム画像送信を主とした規格であり，超低ビットレートでの映像送信を実現する。また，現行のメタル回線ではなく，光ファイバーによる回線ではMPEG程度は楽勝だ。現状で見ても，同軸ケーブルによるケーブルテレビではMPEG2画像配信とデジタル専用線によるインターネット接続を軽くこなしている。ちょっと前まではMPEG2の遠隔ビデオサーバなど非常識きわまりないものだったのだ。ここにゲームサーバが入るだけでもなにが起きるかわからない。世界の常識は日々変化しているのだ。

■まとめ

以上のように，通信ゲームに必要なものは，その物理的な構成によって大きく異なるのだが，リアルタイムアプリケーションの処理という意味では，通常のゲームを制作するうえでの制約や，問題解決手段となんら変わりはない。

むしろ，通信ゲームの根幹をなす，ゲームシステムそのものが，どのような構成であるかといったことや，通信そのものが前提となったゲームの構造で，初めて独自性を発生するものである。通信ゲームは，これからますますいろいろな分野のものが登場し，ますます一般化するであろう。また，データ配信や，インフラの整備によって，もっと投資対象としてのリスクが少なくなってくるので，その結果，まったく新しい，通信がないと成立しないゲームシステムや，新しいゲームの楽しみ方が生まれることも考えられる。

ただ，やはりいえることは，もともと通信しない条件で面白い要素や魅力を持っているかどうかと，どの程度まで丁寧に作り込まれたかどうかが商品価値として判断材料となるといった点は，昔から変わらない。

当初は通信ゲームであるというだけで，話題を集めるものも出てくるかもしれない。しかし，通信できることが当たり前になったとき，そのときこそ本当に面白い，ユーザーに喜ばれる商品が評価されるのである。

DirectPlayを使った対戦ゲーム

菊地 功 *Kikuchi Isawo*

Windows上でのネットワークゲームといえばDirectXの一部でもあるDirectPlayを使用するのが手っ取り早いでしょう。例によって複雑な処理をまとめて面倒見てくれます。難しいとは全部やってくれるので，ネットワークの知識がなくても大丈夫？

ネットワーク……確かにこの先必須となる分野であることは間違いないのだが，初心者から見ると，なかなかどうして難解で，敷居の高い領域だ。筆者も先日，ファミレスでお冷がほしかったのでウェイトレスにアイコンタクトを取ろうとしたところ，ネゴシエーションに失敗してしまった。どうやらプロトコルが違ったらしい。奥が深い。しょうがないのでテーブルに据え付けてあるボタンを押したのだが，これが規格化ということなのだろう。普通ならば，ウェイトレスのいる場所と向きを確認し，ウェイトレスに聞こえるくらいの，それでいてほかの客の迷惑にならない程度の音量で発声しなければならない。しかも状況は刻一刻と変化するわけだから，なかなか大変だ。それが，窓がたくさんあるファミレスでは，テーブルに据え付けてある〝DirectPlay〟と書かれたインタフェイスボタンを押すだけで，用件を聞きにきてくれる。便利だ。

筆者はネットワークに関してはほとんど無知だ。どのくらい無知かというと，家庭内LANを組んで，自分でプロバイダと契約して，ホームページを開いている程度の無知である。だからこの本を読んでいる人の平均くらいの無知さだと思う。しかしである。なんとDirectXにより「おまえらみたいな無知でも，ネットワークゲームが作れるように，簡単なインタフェイスを用意しといてやる。ありがたく頂戴するように」という御触れとともに，DirectPlayというインタフェイスを賜った。やってくれるぜビル。くれたもんは使わにゃ損である。

くどいようだが，筆者はネットワークに関しては無知である。したがって，DirectPlayについて，詳しく説明することは無理である。しかし，どうすればネットワークゲームが作れるかを説明することはできる。ポイントさえ押さえてしまえば，どうということはないのだ。なぜなら，ネットワークとはいっても，やることは基本的に，

送る
受ける

の2つだけだからだ。

それ以外の初期化がうんぬんとか，プロトコルがかんぬんとかいうことは，DirectPlay（のサンプルソース）に任せてしまえばいい。

たとえばだ，いま画面に1台の車が表示されている。その車はカーソルキーでびゅんびゅん疾走するわけだが，1台では寂しいので，もう1台表示するとしよう。ただし，今度は入力をカーソルキーの代わりにDirectPlayで送られてくるデータにすげ替えてしまう。ついでにこちらのカーソルキーの入力を相手にも送ってしまえば，ほら，もうネットワークゲーム（？）が完成した。あとはゲームの作り込み次第だ。バリバリのネットワーク対戦格ゲーでも作って，社内LANでみんなに見せびらかせば，羨望の眼差しを浴びるとともに，上司からも大目玉を頂けること間違いなしだ。

■基礎知識

「ネットワークの知識は不要」といったが（あれ？いってないか？），DirectPlayの基礎知識としてほんのちょっとだけ知っておくことがある。たいして量はないから，心して読むように。

- DirectPlayを使うならDirectX6以上を使え

X5以前とは互換性が怪しいらしい。

- DirectPlay4AとDPlayLobby3Aをダイナミックに接続せよ

DirectPlay4AがX6で追加されたDirectPlayの最新インタフェイスで，最後のAはANSIバージョンであることを示す（ない場合はUnicode）。

DPlayLobby3Aというのは，ロビーと呼ばれ，まあホテルのロビーみたいに，やってきたプレイヤーを，現在イベントが行われている（つまりゲーム中の）部屋まで案内してくれるというものだ。ロビーサーバをアプリケーションと切り離すこともできるが，筆者の美的感覚からいって，くっつけてしまうことをおすすめする。それ以上深くは考えるまじ（ひでぇ説明だなぁ）。

「ロビーはどうしてる？」

と聞かれたら（そんなこと聞く奴いないか），

「3Aをダイナミックに接続してる」

と答えれば，「こいつわかってやがんな」と思ってもらえるかもしれない。ただし，それ以上深く追及されると危険なので，急いでいるふりをして，早々にその場を立ち去ろう。

■ピアトゥピアセッションとクライアント/サーバセッション

セッションというのは，まあゲームがプレイされる部屋みたいなニュアンスだ。それぞれの部屋で，プレイヤー同士がどのようにメッセージを送ることができるかを規定できる。ピアトゥピア（Peer to Peer）というのは，まず誰かひとりがセッションを起こし，その人がサーバプレイヤーとなって，ほかのプレイヤーの参加を管理するが，ゲームが始まってしまえば全員が対等である。そのセッションに参加したプレイヤーの誰からも，誰に対しても，自由にメッセージを送受信することができる。つまり，普通の対戦ゲームというの

は，たいていはこの形式だ。

それに対し，クライアント/サーバセッションというのは，サーバを立ち上げた絶対的なプレイヤーがいて，サーバプレイヤーはクライアントプレイヤーの誰とでも通信できるが，クライアントプレイヤーはサーバプレイヤーとしか通信ができない。イメージとしては，ギャンブルゲームで，サーバプレイヤーはディーラー，クライアントプレイヤーは賭けをしにきた客といった感じだろうか。

ただ，必ずしもこれに従う必要はなく，たとえばクライアント/サーバセッションでクライアントがサーバプレイヤーに対して，「ほかのクライアントプレイヤーの情報を頂戴」とおねだりして，見かけ上はピアトゥピアセッションのように振る舞うこともできるし，逆もまたしかりである。結局のところ，好みの問題ということになるが，プレイヤー数が増えると，ピアトゥピアセッションのほうがネットワーク負荷が大きくなる（もちろん送るデータ量にもよる）。その場合には，ネームサーバを立てて，負荷を減らすという方法もある。

■DirectPlayプロトコル

保証のないサービスプロバイダで，保証つきメッセージングをサポートするプロトコル（意味がわからん？）。サービスプロバイダっていうのは，セッションを立ち上げたプレイヤーのコンピュータのことで，そのときに次のプロトコルを選択できる。

- TCP/IP
- IPX
- モデムトゥモデム
- シリアルリンク

このうち，標準で保証つきメッセージングをサポートしているのは，TCP/IPだけである。保証がない場合には，なにかの拍子にパケットがドロップして，相手に届かない可能性もある。そのような場合に，DirectPlayプロトコルを実装するためのフラグをつけておくだけで，DirectPlayがちゃんと保証してくれるのだ（TCP/IP+DirectPlayプロトコルも可能）。なにか至れり尽くせりだなぁ。

■問題は「同時性」

細かい部分はまたあとでおいおいやるとして，ゲーム側から見たネットワークによる制限を考えてみよう。まずは転送できるデータ量が少ないこと。たまにネットワーク対戦ゲームが，画面に表示されたグラフィックを1フレームずつすべてを相手に送っていると思っている人がいるらしいのだが，まさか本誌を読んでいる人に限ってそんな人はおるまい。

主にピアトゥピア接続の場合には，それぞれのマシンで同時に同じゲームを走らせているので，それぞれに入力値だけを与えて，同じ計算を行って整合性を保つことになる。

ネットワークの負荷を抑えるため，極力データ量は減らすべきだ。すでに先ほど述べてしまったが，理屈ではキー入力データだけを送って，アクションはそれぞれのクライアントマシンで処理するようにすれば，1回の送信はほんの数バイトで済む。

理屈では。というのは，ネットワークでデータを相手に送るとき，かかる時間はまちまちだからだ。なにも考えずにつなぐと，とんでもないことになるのは簡単に予想できるだろう。たとえば，プレイヤーAの車が上に向かって走っているとしよう。ちょうど画面の真ん中でハンドルを右に切った，しかしその瞬間のネットワークのトラフィックが大きく，相手にその操作が届くのが一瞬遅れてしまった。すると，プレイヤーBの画面では，Aは画面の中央よりも少し上で右に曲がったと判断されてしまう。それが積もっていけば，あっという間に誤差が誤差を生み，とてもゲームどころではなくなってしまう。

こんなこともありうる。AがBに向かってミサ

COLUMN

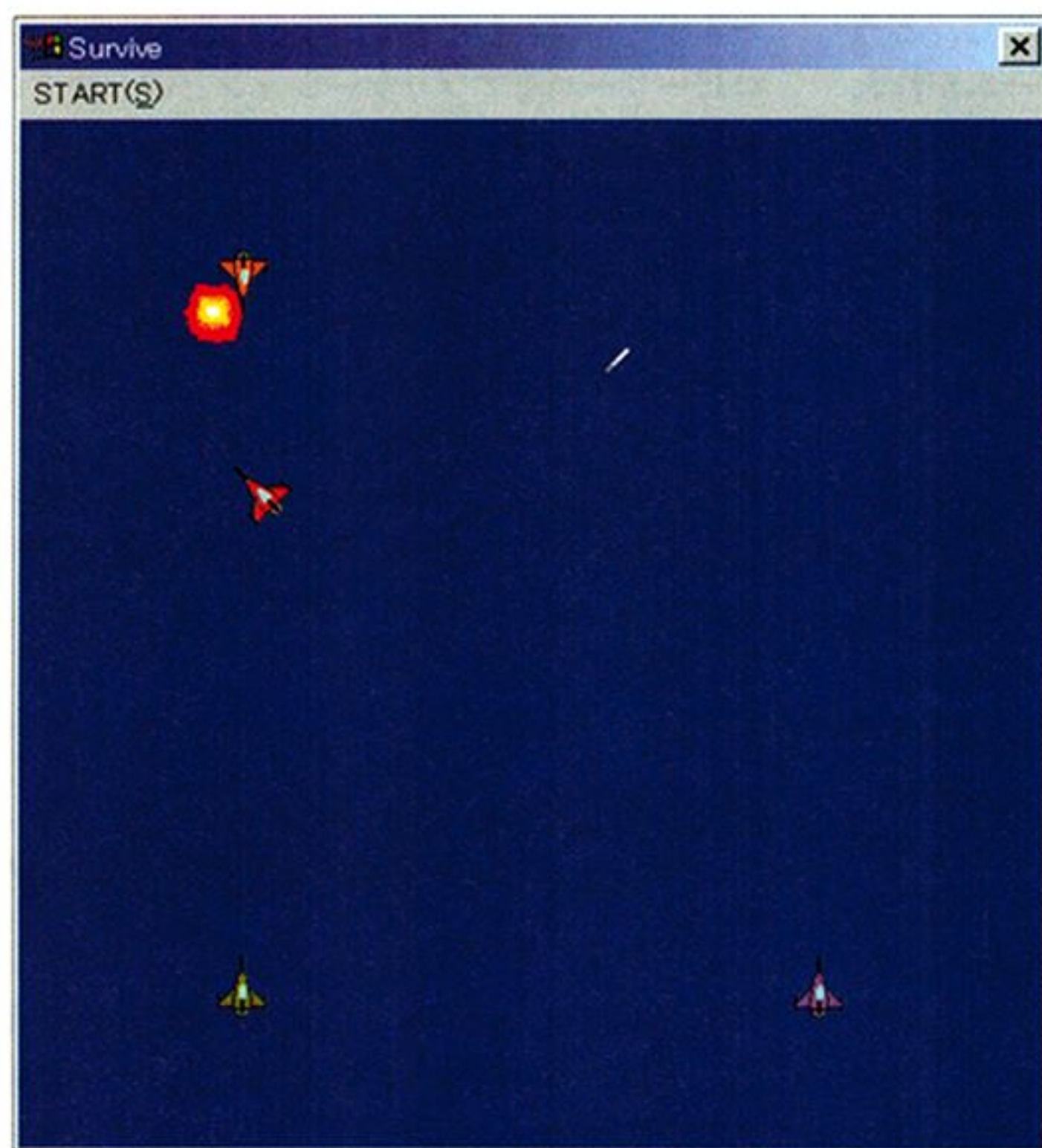



図1　Surviveのゲーム画面

Surviveは最大12機の戦闘機がステージ内を飛び回り，ミサイルを発射して他人の戦闘機を撃ち落とすサバイバルゲームである（図1）。ステージは上下および左右がつながっており，右端から外へ出たものは左から，上から出たものは下から現れる。ミサイルも同じである。ただし，ミサイルには射程があり，だいたいステージ1面分くらいの距離を進むと自然消滅する。戦闘機は放っておくと直進し続け，カーソルキーの左右で旋回できるが，スピードの増減や停止はできない。スペースキーでミサイルを発射できる。各機が画面内で表示できるミサイルは3発までである。つまり，3発発射してしまうと，それが敵に当たって消滅するか，射程により自然消滅するまで次弾は発射できない。ミサイル同士の当たり判定はないが，自分の撃ったミサイルと自機との当たり判定は行っており，当たるともちろん死ぬ。DirectPlay実装前のSurviveテスト版では戦闘機同士の当たり判定は行っていないが，実装後のDPSurviveは戦闘機同士で衝突すると対消滅する。

とりあえずゲームスタートすると，12機の戦闘機が現れる。しかし，操縦できるのは赤い戦闘機1機のみで，ほかの戦闘機はただまっすぐ進んでいるだけである。DirectPlayを実装する段階で，ほかのプレイヤーから送られてきたメッセージからキーデータを抽出し，ほかの戦闘機をリモート操作できるようにする。ゲーム終了条件は生存者が1名以下になることだ。ただし，DirectPlay実装前は操作できる戦闘機が1機だけなので，自分のミサイルで自滅してしまうとゲームは永遠に終わらない。というわけで，敵を倒すよりは自滅しないということに集中する必要があるかもしれない。とりあえず，弾を撃ったあとは急旋回しない。

また，Dinputに入ったサンプルは本文のほうでも述べた，バッファによる遅延のシミュレーションでもある。キーから入力されたデータをリングバッファに格納し，わざと遅延させている。ゲームスピードは10fps，バッファカウントは3に設定，つまり遅延は0.2秒である。この辺りはsurvive.h内のFRAMERATEおよびBUFFERCOUNTでdefineしてあるので，適当にいじってリビルドし，操作感覚を確かめてみるとよいだろう。

このゲームはDirectDrawを使って作られている。Windowモードでカラーキーを使ってキャラクターを擬似スプライト表示しているだけなので，DirectDrawとしては特に難しいことはしていない。この辺りのことは，DirectDrawのサンプルソースddutil.cppおよびddutil.hを引っ張ってくればお手軽だ。

ただし，あることをやろうとして愕然とした。「色違いの戦闘機」である。こういう場合，グラフィックパターンはひとつだけ用意して，あとはパレットの変更により色違いにするのが普通だ。当たり前だ。それしか考えられない。にもかかわらず，DirectDrawはそれがまったくなんにもさっぱり考えられていない（DirectDrawのいうところのパレットはまた別次元の話である）。信じられん。ほんとにこれはゲームのシステムなのだろうか？　こんなにありがちで当たり前で絶対不可欠な機能がないなんて。あいつらはカラーキーがあれば「スプライト」だと思い込んでるのだろうか。そのうち非矩形ウィンドウもスプライトウィンドウとかいいだすのではあるまいか。四角くないものはすべてスプライトか。それはスプライトに対する冒涜だぁ！　どっか～ん!!

なにも考えずにこれを解決しようとすると，リソースとして12機分の色違いの画像を持たせることになる。どうせCD-ROMに載せるのだからファイルサイズが多少でかくなるのはたいした問題ではないのだが，筆者の美的感覚がそれを許さない。いっそスプライトの名を汚したDirectDrawなど見限って，GDIでやろうかとも思った。幸いちょうどほかのプログラムで作った，ビットマップとパレットを切り離す関数があったし。しかし，ネットワーク以外に重い部分をさらに引きずるのもイヤだし，DirectPlayも使うんだからということで，その関数を流用してDirceDrawを使うことにした。デバイスコンテキ

イルをぶっ放した。Aの画面ではすでにBに命中し，クラッシュしている。しかし，Bの画面ではまだ命中しておらず，Bはミサイルに気がついて間一髪かわし，カウンターでAに手榴弾を投げて吹き飛ばした。Bからすれば，自分はかわしたつもりなのになぜか死んでいるし，Aにしてみれば，幽霊に殺されたというパラドックスが発生する。

この2つは，ともに「同時性」に起因する問題である。ネットワークでデータを送る分，お互いのマシンで自分よりも相手のほうが時間が遅れているだけでなく，時間のムラにもなってしまうからだ。これを解決するいちばん手っ取り早い方法は，「同期させる」ことだ(ここでいう「同期」はDirect Playのヘルプで説明されている「同期」とは異なる)。つまり，相手のデータが送られてくるまでゲーム時間を進めずに，お互いのデータが揃って初めて1フレームを処理する。時間はゲームがスタートしてからのフレーム数で計算される。いってみれば，リアルタイムなターン制みたいなものだ。

そうすれば，あるフレーム数で表される時間の，お互いの位置に誤差が生じたり，状況が異なったりすることは，昔流行ったバグ入りPentiumでもない限りありえない。また，同期させれば「回線が太いほうが有利」という理不尽なこともない。

ただし，このように同期させてしまうと，回線の状態がモロにゲームのスピードに響いてくる。ネットワークが十分速いときには，ゲーム側で設定したフレームレートでスムーズに進行できるが，トラフィックが大きくなってくると，相手のデータが送られてくるまで止まってしまうため，ちょくちょく引っかかったような感じになるだろう。ひとりだけでも遅いモデムや回線状態の悪いプレイヤーがいたらゲーム全体の速度がガクッと落ちる。

とはいえ，同期を取らないシステムだとゲームの構成が難しくなる。

たとえば，強力なゲームサーバを立てて，ゲーム本体をその内部だけで進行させるなら同期は必要ない。クライアントから入力を集めて，サーバはサーバの時間で進行し，クライアントには結果を返すという形式だ。この方式の場合，やり取りするデータ量が多くなるのと，通信回線の質やクライアントマシンの速度でゲームに有利不利が生じるのはしかたないところだ。クライアントとサーバの処理分担を，どの程度ずつに振り分けるかが問題になってくるだろう。これはゲーム内容によってまったく異なってくると思われる。

ということで，ここではやはりピアトゥピアセッションを中心に考える。インターネットを使用して接続を行っていると，通信速度は保証されないし，パケットそれぞれでの時間ムラがもろにゲーム進行に影響してくるようになる。たとえば，たいていは大丈夫なのだが，パケット10個に1個くらいの割合で遅れることのある回線状況だと，だいたい10フレームに1回ゲーム進行がガタつくようになる。これがユーザーの数だけ集積されるとギッタンバッタンした速度でしか進行しないのは目に見えている。そこで，バッファリングを行い，データ受信の平滑化を図る。

つまり，送られてきた(およびキーボードから入力された)データをまずリングバッファに蓄え，ゲーム側は古いデータから使っていく。仮に瞬間的にネットワークのトラフィックが大きくなり，データが滞っても，バッファが空になるまでは，ゲームはフリーズすることはない。

しかし，この方法だとキーを入力してから実際にそれが画面に反映されるまで，タイムラグが生じてしまう。人間は0.1秒を超えると，時間のずれを認識すると聞いた覚えがあるが，ネットワークゲームだから0.2～0.3秒くらいは許されるんじゃないだろうか(別に根拠はない)。つまり，0.1秒単位で進行するゲームならば，バッファは3～4段ということになる(バッファ1段は遅延なしと同じ)。

■DirectPlayの実装

もととなるゲームのシステムSurviveについて

対戦型生き残りシューティングゲームSurviveについて

ストにパレットを設定し，DirectDrawサーフェスにBitBlt，それを12回繰り返す。結局メモリ上にはサーフェスが12枚できてしまうが，リソースの段階から12枚あるよりはなんぼかました。ちなみに最大12人対戦というのは，16色パレットのうち共通カラーが4色，残り12色がボディカラーとして残ったからというだけの安易な判断である。

戦闘機の向きは16方向である。16方向くらいあると，座標に何ビットか下駄を履かせないと，向きによるスピードのばらつきが目立つかと思ったが，そうでもないようだったのでやめた。たとえばこういうことだ。スピードが4で右向きに飛ぶ戦闘機，この場合x方向の増分は4，y方向の増分は0，すると移動量は4ぴったり，当たり前だ。これが30度だけ上を向いたとする。すると，x方向は4×sin30°＝2，y方向は4×cos30°＝3.46…四捨五入して3となり，移動量はsqrt(2^2+3^2)＝3.60……と，約1割減速してしまう。それとともに，30度上に向かって飛んでほしいのに，実際にはatan2/3＝33.69……となり，約3.7度向きがずれてしまう。こういった場合，小数点演算よりも何ビットか下駄を履かせるのが一般的だ(と思う)。スピードや座標を4ビットないしは8ビットくらいシフトアップした値で演算し，実際の画面への描画時に座標値をシフトダウンして描画座標を求める（固定小数点演算ともいえる)。スピードの値が大きければ誤差も少ないわけで，このSurviveはスピードが16，DPSurviveは調整して10としたが，これくらいならばそれほどひどい誤差は出ないようなので，そうしなかったわけだ。

なお，移動量については，いちいち三角関数で求めるのではなく，あらかじめ16方向の移動量を求め，それを配列に保存してある。まあ最近のCPUではそんなことをしてこっちのほうが速いといったところで，絶対にわからないが，気分の問題だ。この移動量はミサイルでも使われている。ミサイルは戦闘機の2倍のスピードということで，1フレームで2回加算しているが，当たり判定をそのあとだけにすると，その移動によってちょうど戦闘機をまたぐ形になってしまったとき，ミサイルのすり抜け現象が起こる。そこで，1回加算した時点でも当たり判定を行っている。

各機の情報は，

```
struct PLAYER {
        BOOL life;
        int direct, x, y;
        int mdirect[3], mx[3], my[3];
        BOOL mlife[3];
        int time;
} Player[MAXPLAYERS];
```

という構造体配列に入っている。lifeはまだ生存しているかどうか，directは向きを示し，0が真上，右回りに16段階，x,yは座標である。mdirect,mx,myは弾3発分の向きと座標である。mlifeは弾が画面に表示されているかどうかであるが，実際には弾の余命が入っている。発射されたときに寿命が入れられ，1フレームごとにデクリメント，0になると消滅というわけだ。timeは死んでしまったときのフレーム数，つまり生き残っていた時間が入れられる。

あとの細かい部分は，ちょっとここでは説明しきれないので，ソースを見て理解してもらいたい。なにかの参考にするには過激なほどコメントがないが，そこはそれ，苦しんで理解したほうが身につくというものだ。だってほら，若いころの努力は買ってでもしろっていうじゃない。この本を読んでる人はもう本は買ったわけだから，あと必要なのは「努力」だね（なんか違う)。

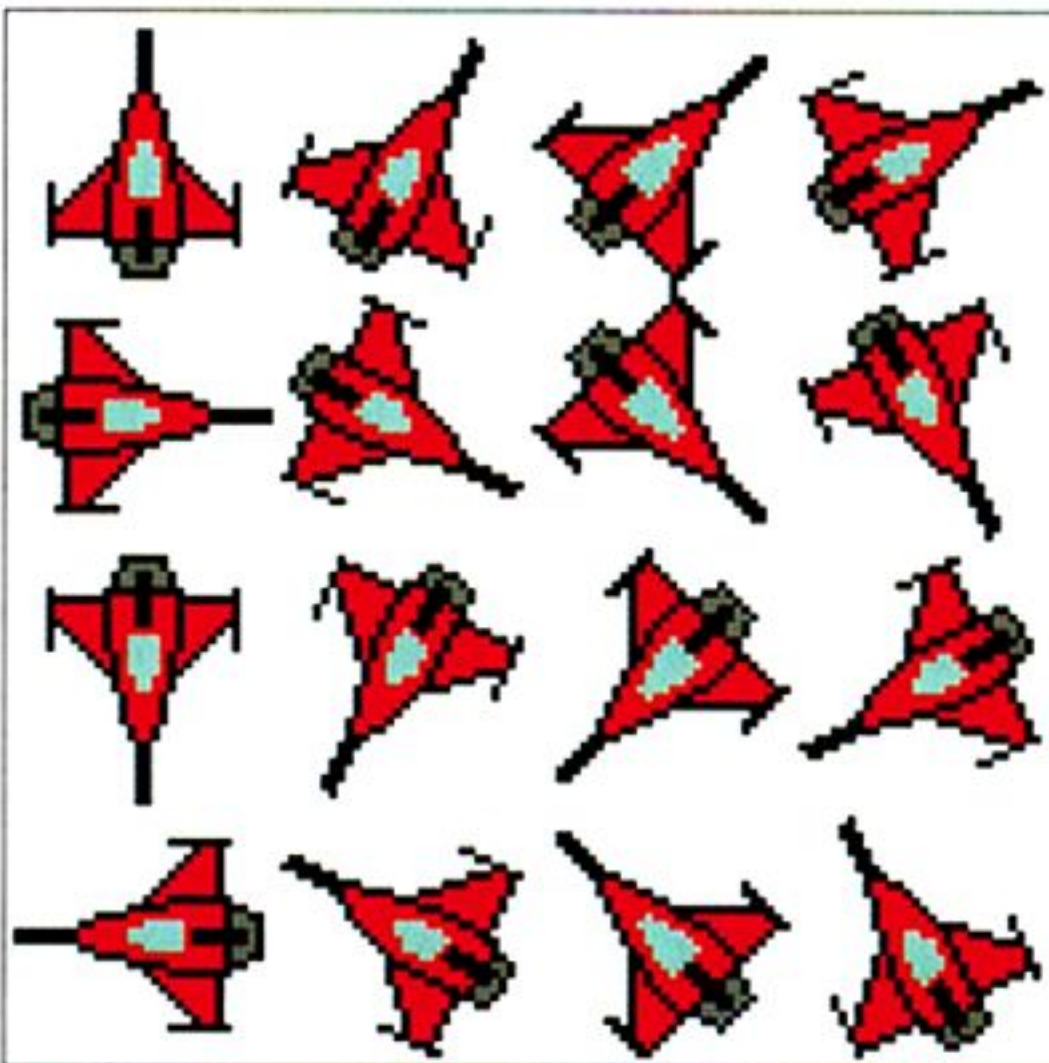



図2　飛行機の基本パターン

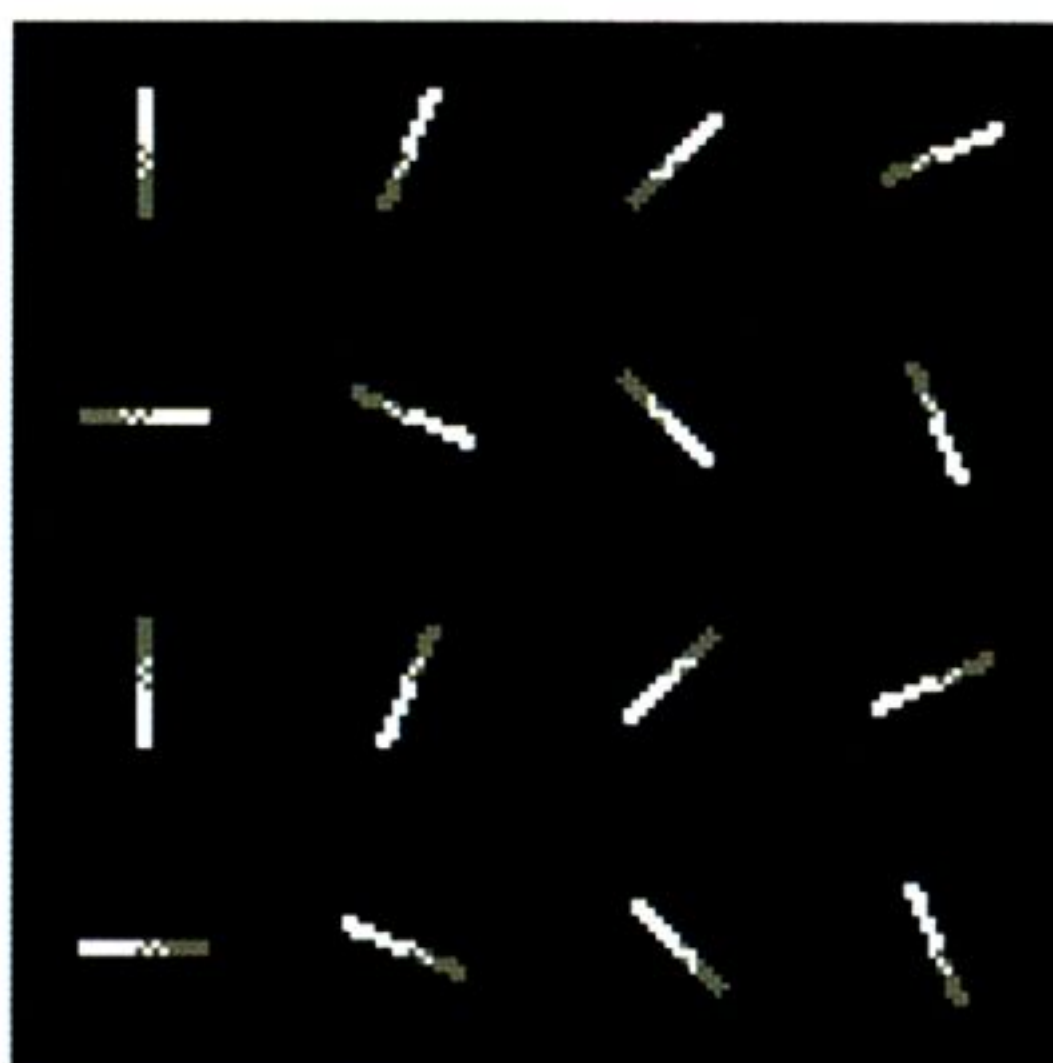



図3　ミサイルもこうやって先に作っておく

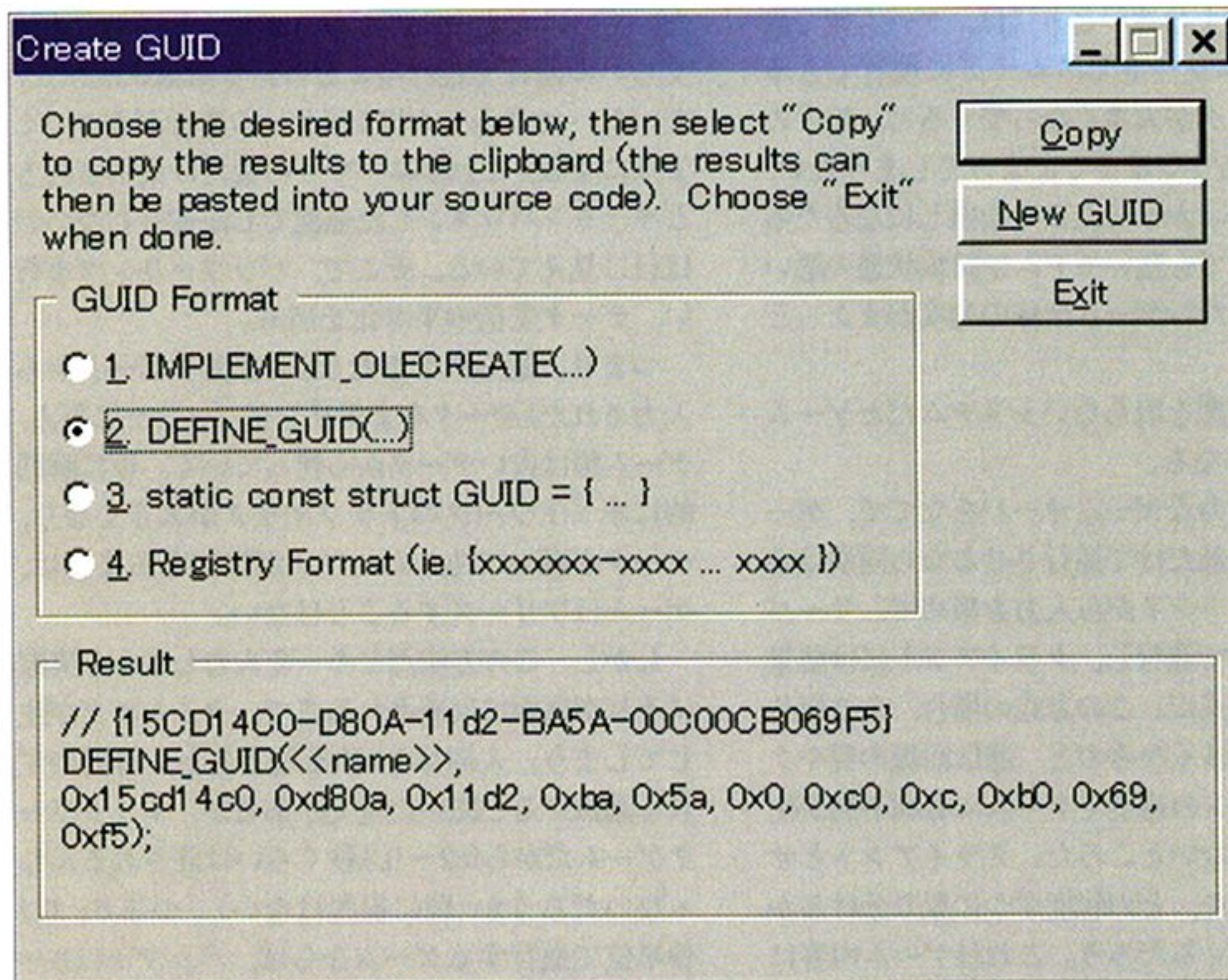



図1　GUIDGEN.EXEで面倒なことはやってくれる

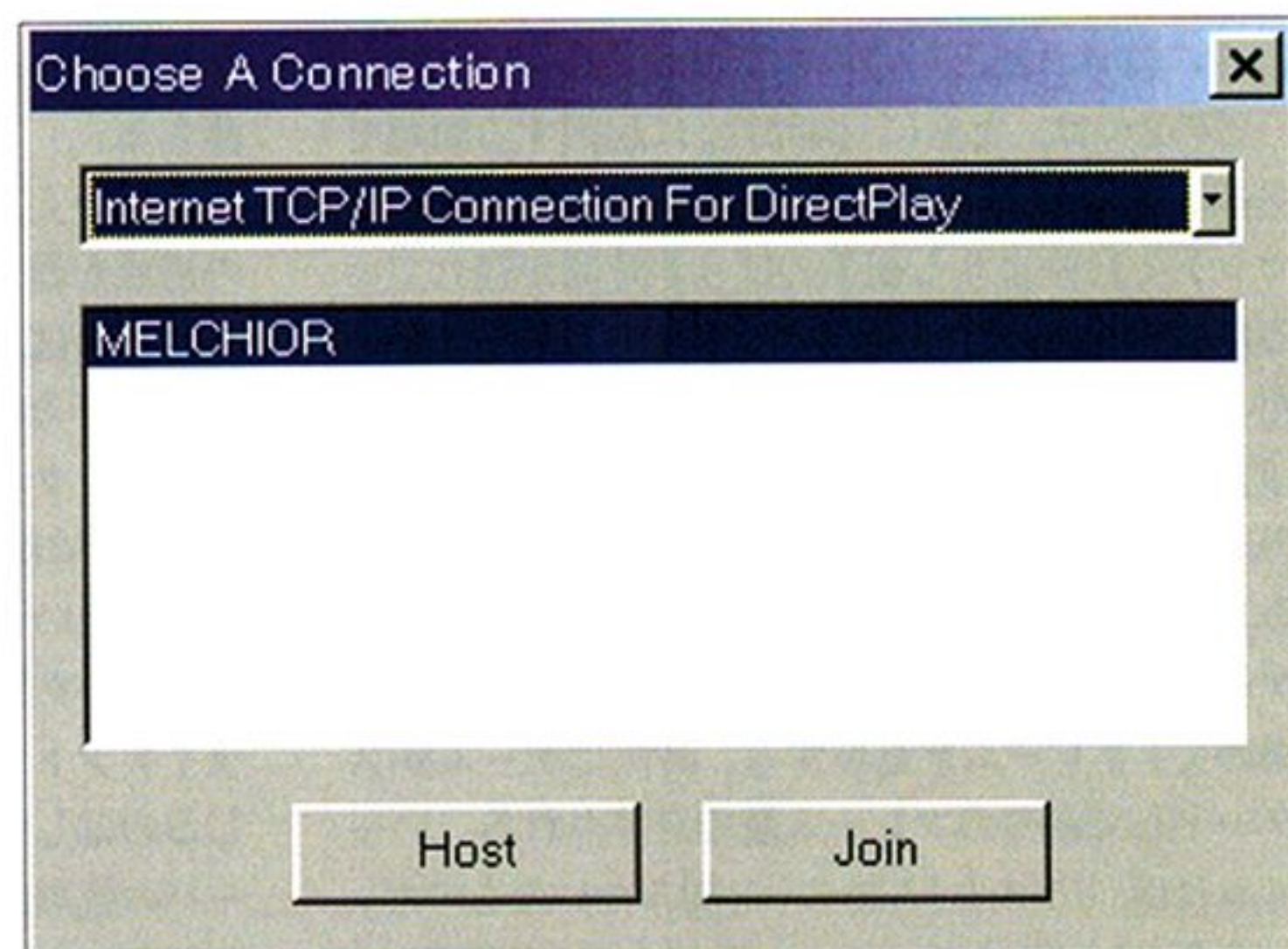



図2　接続するサーバを選択

はコラムを参照してもらうとして，それにDirect Playを実装していこう。DirectX SDKに付属していたDirectPlayサンプルDPChat辺りのソースを引っ張ってくるのが手っ取り早い。

必要なファイルは，C:¥mssdk¥samples¥Multimedia¥DPlay¥src¥DPChat（DirectX SDKをC:¥mssdkにインストールした場合）の中のdialog.cppとlobby.cpp，それとdpchat.hも大部分がそのまま使えるので，適当に名前を変えてプロジェクトに加える（今回は単にdp.hとした）。

あと，dpchat.cpp内のSetupConnection()からHandleSystemMessage()までの5つの関数と，いちばん最後のGetDirectPlayErrStr()関数を，ごっそりSurvive.cpp（メインソース）内に移し，dpchat.rcに入っている2つのダイアログリソースも持ってくる。とりあえずこれで下準備は完了だ。

lobby.cppというのがロビー部，dialog.cppはロビーがセッション表示に使うダイアログだ。dpchat.cppから移植した部分がロビーを呼んで接続を確立し，メッセージの送受信を行う部分である。いじるのは主にこの送受信の部分だ。ここで自分のキーデータを送信し，ほかのプレイヤーからのキーデータを受信しバッファに入れる。また，新しいプレイヤーの参加メッセージや，ゲームの開始メッセージの処理なども入る。

ところで，先に「基礎知識」で説明した部分を確認しておこう。DirectX6 SDKのサンプルだからDX6はOK，DPlayLobby3Aもダイナミックに接続されている（というか，接続されているサンプルを選んだ）。Surviveの場合はピアトゥピアセッションが向いており，DPChatもピアトゥピアセッションで作られている。

最後のDirectPlayプロトコルについては，少々文字が落ちてもあまり支障はないため，DPChatは保証なしでやっているが，Surviveでは致命的であるので，これはDirectPlayプロトコル付きに変更しておこう。とはいえ，前述のようにフラグをひとつつけるだけだ。セッションを作るときのIDirectPlay4A::Open()メソッドの第1引数DPSESSIONDESC2構造体の中で設定する。dialog.cppの366行目，

```
sessionDesc.dwFlags = DPSESSION_
MIGRATEHOST | DPSESSION_KEEPALIVE;
```

にDPSESSION_DIRECTPLAYPROTOCOLも加えてやればよい。

では本体に手をつけよう。dpchat.cppから移植したSetupConnection()関数は，要するにDirectPlayやロビーの初期化，およびセッションの作成・参加部分であるので，アプリケーションの初期化部分から呼び出しておけばいい。ShutdownConnection()はその反対で後始末であるので，アプリケーションの終了時に呼ぶ。

ReceiveThread()はメッセージが送られてくるスレッド関数であるのでユーザーが直接コールする必要はなく，ReceiveMessage()はReceiveThread()がメッセージを処理するために呼ぶ関数だ。ここまでは内容を変更する必要がない。

カスタマイズが必要なのはその次だ。HandleApplicationMessage()とHandleSystemMessage()は，前者はアプリケーション固有のメッセージを，後者はDirectPlayのシステムメッセージを処理するため，ReceiveMessage()が呼び出す関数だ。この辺りは，Windowsのメッセージ処理とよく似ている。実際そういったメッセージ処理を手動で（MFCを使わずに）やったことのある人ならば，すぐに理解できるだろう。そうでない人は，なんとか理解してほしい。

DirectPlayのメッセージは，必ず次の構造体で送られてくる。

```
DPtypedef struct {
    DWORD dwType;
} DPMSG_GENERIC;
```

中身はDWORDが1個しかない。だが送られてくるのはこれだけではない。後ろにデータはくっついている。ただ，なにがどれだけくっついているかはわからない。それはdwTypeで判断するのだ。dwTypeはわかりやすくいえば，メッセージIDのことである。だからもしそれがDPSYS_CREATEPLAYERORGROUP（新たなプレイヤーまたはグループが加わったときのシステムメッセージ）だったら，ああ，ここで渡されたデータはDPMSG_CREATEPLAYERORGROUP構造体なんだなと判断して，DPMSG_GENERICからDPMSG_CREATEPLAYERORGROUPにキャストして，参照することができる。

自分でメッセージを作った場合も，そのメッセージのデータ構造を自分で決めておけばよい。たとえば，キーから入力されたデータを送受信するメッセージをAPPMSG_KEYOPERATION，そのデータ構造を，

```
typedef struct {
        DWORD   dwType;
        BYTE    keydata;
} MSG_KEYOP;
```

として定めておく。送信するときはこの構造体をグローバルメモリに確保し，dwTypeにAPPMSG_KEYOPERATIONを，keydataにキーデータを入れ，IDirectPlay4A::SendEx()で全員に配信する。受け取り側は，やってきたメッセージのDPMSG_GENERIC構造体のdwTypeを見て，もしAPPMSG_KEYOPERATIONだったならば，それをMSG_KEYOPにキャストすることで，keydataを抽出できる。

すばらしい。ネットワーク通信が単なるメッセージのやり取りになってしまった。実際には，送信したデータの順番と受信したデータの順番が入れ替わる場合も考えられるので，キーデータの上

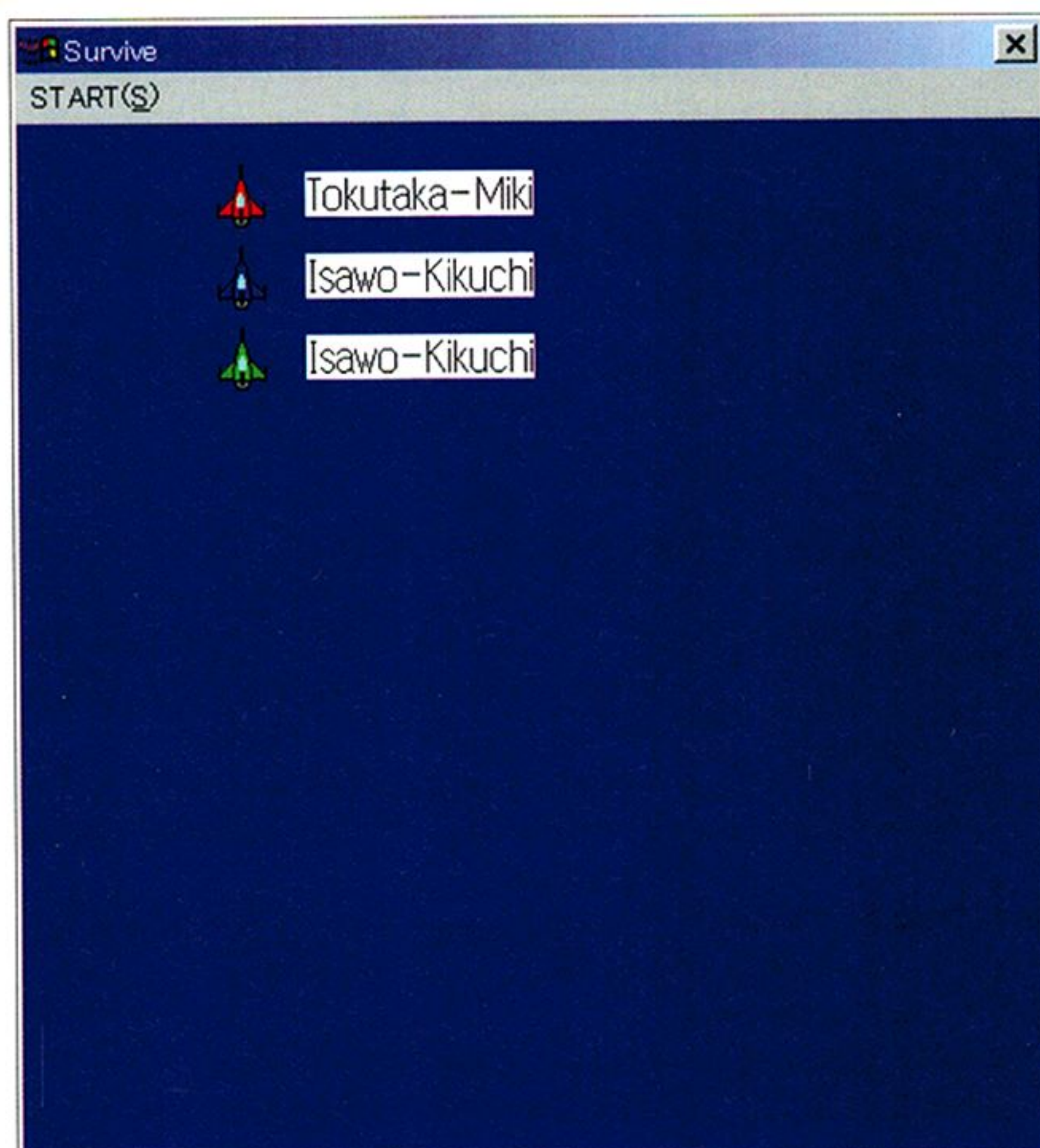



図3 ゲームにログインするとプレイヤー一覧に登録される

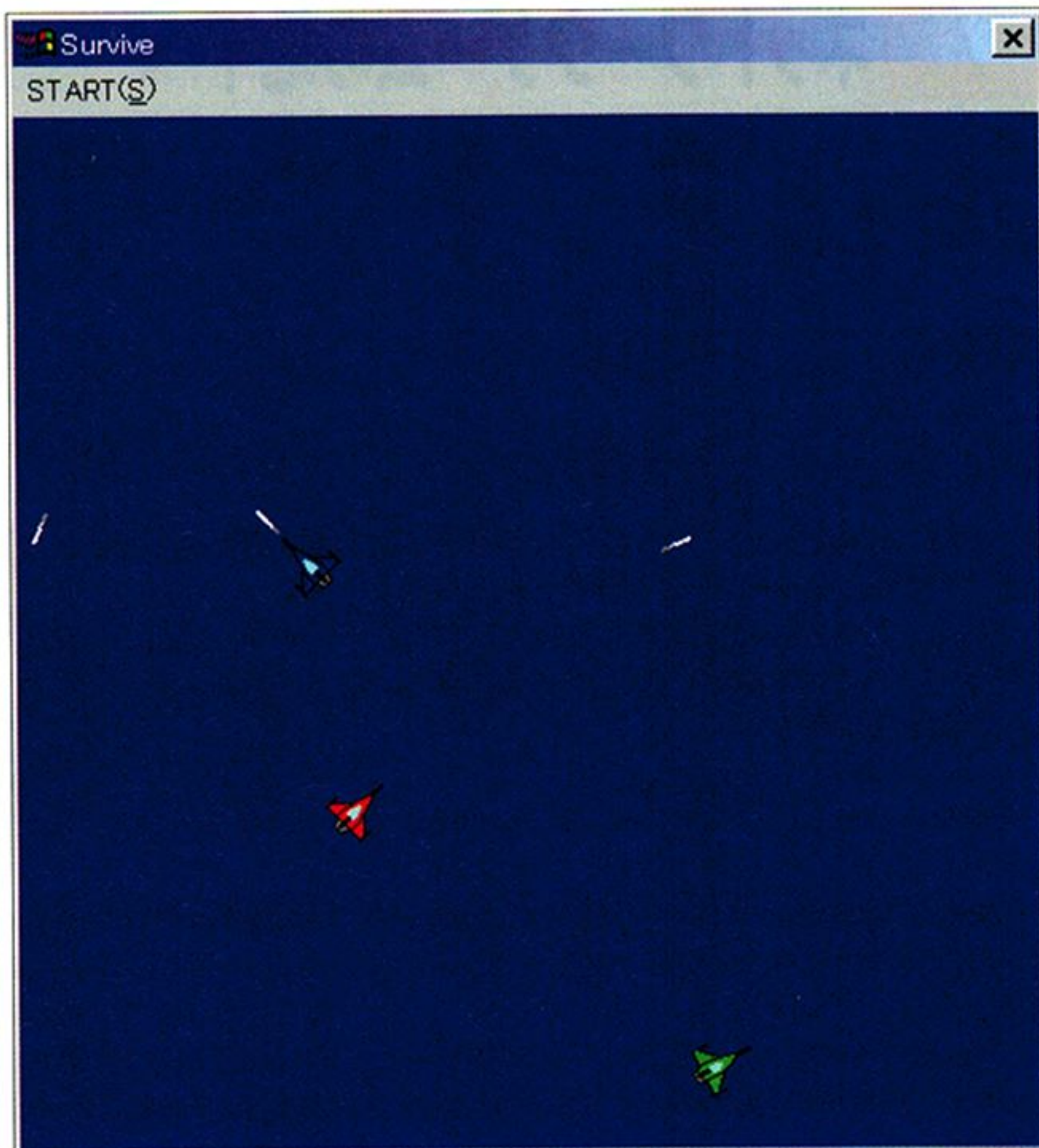



図4 あとは撃ちまくるだけ

位4ビットにはキーを格納すべきバッファ番号を入れている。

また，キーデータ送受信メッセージIDはひとつではなく，12人分用意してある。なぜなら，誰のものかを示すID変数を用意する必要がなく，データ量が少なく済むからだ。

ただ，ここでちょっと問題があった。同じセッション内のプレイヤー間で，時間のずれは最大でバッファ数+1生じる。進んでいる側のマシンでバッファが空になってしまうので，それ以上ずれは広がらないのだが，問題は遅れている側だ。ちょうどひと回り先のデータまで送られてくるので，なにも考えずにバッファに上書きしてしまうと，これから使われるはずだったデータが，1回り先のデータとすり代わってしまう。そのため，ダブルバッファリングを行い，もし格納すべきバッファ内のデータが未使用の場合は予備バッファに蓄え，データを使用したときに予備バッファからデータを落とし込む方法を取っている(編注：素直に止めれば済みそうなもんだが……)。

■GUID

ロビーは，現在動いていて接続できるセッションを検索するときに，GUIDで判断している。GUIDとはアプリケーションを判別するユニークな識別子である。DirectPlayを使っているからといって，まったく違うゲームを接続しても，ゲームにはならないからだ。それはそうだろう。お互いが同じメッセージを共有し，同じルールの下で動かなければ，正しいゲームは成立しない。

DPChatのGUIDはdp(chat).h内で宣言されているのだが，上記の理由でそれをそのまま流用してはならず，新しいアプリケーションを作る場合は，新しいGUIDを生成しなければならない。GUIDを生成するツールはVisualStudioについている。Commonフォルダに入っているGUIDGEN.EXEがそれだ(図1)。起動するとすでにGUID自体は作られているので，GUIDフォーマットを2番にしてCopyボタンを押す。あとはソース中にペーストするだけだ。

だが，これをするとdxguid.libが邪魔だといいだす。そこでリンクするライブラリからdxguid.libを外すと，今度はIID_IDirectDraw4がないといいだす始末。なんなんだよ，まったく。しょうがないので，IID_IDirectDraw4の定義部分をddraw.hから引っ張ってきたのだが，きっとなにか方法があるんだろうなぁ。なにかをdefineしておけばいいとかなんとか。ま，いいか，ビルドできたし。

■論より証拠

なんかちゃんと動くらしい。いやあ不思議。詳しい説明はやっぱりできないので，ちゃんと理解したい人はさっぱりコメントのないソースを読んで，どっぷりハマってほしい。とりあえず実行すると，Choose A Connectionとかいってくるので(図2)，「なんだてめぇは，日本語しゃべりやがれ！」ってキレたりせずに，<Select service provider>と書かれたプルダウンメニューを開いてみよう。そこには，TCP/IP，IPX，Modem，Serialの4種類があると思うので，どれかを選ぶ。普通はTCP/IPかIPXだよな。

注意としてはモデムでダイヤルアップ接続だからといってModemを選んだりしないこと。これはモデムで直に2台のマシンをつないだ場合の話だ。TCP/IPの場合は，ホスト名かIPアドレスを聞いてくる。ホストプレイヤーになるのであれば，なにも入れずにキャンセルし，Hostボタンを押せばいいのだが，ここはぜひ編集部で立てたサーバに接続してみてもらいたい。ホスト名はxlabs.pub.softbank.co.jpだ。

待つこと数秒，すでにセッションが起きていたら一覧が表示されるので，そこから接続したいセッションを選んでjoinする。ゲーム中なら途中参加はできないので，しばらく待つ旨のメッセージが表示されるが，ちょっと待っていれば，きっと図3の画面になるだろう。これは現在の参加者のリストだ。プレイヤー名はWindowsのログイン名が使われる。12人まで参加できるが，適当なところで誰かがSTARTボタンを押すと，ゲームスタートだ。あとは心ゆくまで殺しあってほしい。そうそう，プレイヤーがひとりしかいないときにSTARTしてしまうと，始まった途端にゲームオーバーである。それから，ファイアウォールがあると動かないことがあるかもしれない。

しかしまあ，なんだな。できたのはいいが，これだけ説明がいいかげんでコメントも少ないと，参考になるかははなはだ疑問だ。そうだな，こういうのはどうだろう。「筆者のようにネットワーク無知でも，ネットワークゲームを作れることが実証できた」。

ネットワークゲームの地平へ

Javaネットゲー考察 Expert Mission制作編

霧雨 Kiriu

ここではJavaのサーバ/クライアントシステムを使ったゲーム制作について解説します。Java自体はネットワークとの親和性が高いのですが，パフォーマンス評価の結果では，なかなか厳しいのが現状のようです……。

前回紹介した，Javaアプリケーションの記事はいかがだったろうか。我ながらすんげぇ地味な原稿だと書き上げてから改めて反省したのだが，Javaを敬遠してた方の入門参考程度にはなるかと思う。

また，書き上げてから発行までの期間が長かったので，出てから状況変わりまくりというのもキツかった。

で，第2号のテーマがネットワークゲームということなので，昨今流行の通信対戦ゲームをJavaで作成することに挑戦してみた。実行環境が整う方なら，付属のサンプルで実際に接続し，ほかのプレイヤーと対戦をお楽しみいただける。

ネットゲーをする場合のお約束，電話代とプロバイダ代は各自で管理しよう。

ただ遊ぶだけなら別だが，一応，前回の記事を理解していないと，今回の内容は取っつきにくいので，復刊第1号を手元に置いて読み進めるのが吉である。

今回の開発環境はさまざまな事情により前回となんら変わってない。

SONY VAIO PCG-505EX/64
Windows95 OSR2.1
Symantec Visual Cafe for Java 2.0J
JDK1.1.x（1.1.3相当で開発）

もう世間ではWindows98も結構浸透しているのに，アップグレードはどうした？　という声も聞こえそうだし，友人にもさんざいわれた。だが，Windows98を真っ先に入れた人身御供たちの悲惨なトラブル報告は耳にタコができるほど聞いたし，何度かJavaアプリケーションの不具合を目のあたりにしたこともあり，とてもじゃないが仕事にも使う愛機（しかもドライバ関係が面倒なノート）をおいそれと虎穴に入れるような真似はできなかったってゆー事情がある。

Javaをまともに動かすための奮闘記録ならそれもまた記事になりそうなのだが，とりあえずたいていのメソッドは問題ないようなので，それは省略。

そうはいっても，実は別のデスクトップ環境にはWindows98を入れてみたのだが，ご想像どおりイロイロと妙な現象で苦労した。この話をし出すとキリがないし，本誌のテーマと外れるのでこの辺で。

で，せっかくのネットワークゲームなのに，技術の解説から入ってはつまらないかもってことで，今回はどちらから読んでも楽しめるような構成になっている。

・すぐに遊びたい　→　216ページへ
・技術の解説を読む　→　このまま

■第1章：ネットゲーを作るうえでの基礎

Javaはもともとネットワーク上で動作することも前提に作られているので，今回のネットゲーというテーマは得意分野に入っている。

C言語など，現存するほかの言語と比較すると，ネットワークプログラムの作りやすさはトップクラスに入るのではないかと，個人的に思っていたりする。

では手始めに，Javaでネットワークゲームを作る際に最低限必要となる知識を，サクッと頭に入れておこう。

●1-1 IPアドレスとURL

IPアドレスとは，インターネット上で，ネットワークとマシンをユニークに識別するためのデータである。3桁の数字を4つ並べ，各数字間をピリオドで区切ったかたちで表現される。

これをDNSが比較的人間にもわかりやすく変換してくれるのが，最近はすっかりお馴染みになったURLだ。

Javaではjava.net.URLというクラスが用意されている。

●1-2 接続方法とプロトコル

次は接続方法。この形態はいくつか存在するが，もっとも一般的なものとしてC/S（クライアント/サーバ）方式が挙げられるだろう。

これは，サーバとクライアントの2種類のプロセスが動作するというもの。サーバとクライアントは同一ホストでも別でも構わない。

通信プロトコルには，接続型と呼ばれるTCPと，非接続型と呼ばれるUDPの2つがある。

TCPではサーバとクライアント間に専用の経路が作られる。送信したデータは確実かつ順番に受け取られ，データの信頼性は非常に高い。

UDPではサーバ/クライアントの間に専用の経路は作られない。相手先のアドレスを含むパケットで送信されるが，こちらは相手に確実に順番どおりに届くという保証がない。TCPは電話回線に，UDPは郵便に例えられることが多い。多分これで理解しやすくなったのではないかと思うがどうだろう。

ぶっちゃけた話，UDPは滅多に使われることはないようだ。Javaでは両方ともクラスが用意されているが，実際にネットプログラムを作る場合，信頼性のあるTCPを使うだろう。

通常，C/S型のネットワークゲームでは，ユーザーの誰かひとりがサーバになってほかのユーザーを待つ場合と，専用ゲームサーバーが複数のクライアントの制御を行うという場合の2系統がよく見られる。

●1-4 ソケット

次はネットプログラム全般における基礎知識である「ソケット」。

ソケットとは，コンセントや電話回線のプラグのようなものと考えてもらえばいい。

このソケットが実際のプラグのように，通信元と通信先でカチッとはまることで通信が確立する。この概念のおかげで，プログラマは下回りの

リスト1　簡単なサーバプロセスの例

```
import      java.lang.*;
import      java.applet.*;
import      java.net.*;

class ServerConnecter extends Applet{

 // サーバーソケット
 ServerSocket srvSock;
 final int portNum=1357;

 public void ServerConnecter(){
 }

 public void init(){
  try{
   // あいているポートNo.でサーバソケットを開く
   srvSock=new ServerSocket(portNum);
  }
  catch (Exception e){
   System.exit(1);
  }
  // サーバログ用
  System.out.println("Server Process Successfully");
 }

 public void run(){
 // ソケットクラス生成
  while(true){
   try{
    // サーバに接続があるまでひたすら待つ (ここで固まっている)
    Socket Sock=srvSock.accept();
    // クライアント接続管理クラスへ使用可能なソケットを渡す
    if (cliMap.addClient(Sock)==false){
     System.out.println("Client maximum");
    }
    else{
     // 接続に成功。戻り値のSocketはそのままストリームへつなげられる
     System.out.println("Client Connected!");
    }
   }
   catch ( Exception e ){
    System.out.println("Connect failed!");
    System.exit(1);
   }
  }

 public void finalize(){
  // 強制的にソケットを閉じる
  srvSock = null;
 }
}
```

プロトコルの種類や制御に煩わされることなく，上層部のコーディングに専念できるというわけだ。

Javaではjava.net.Socketクラスが接続型(TCP)に，java.net.DatagramSocketクラスが非接続型(UDP)に対応している。

これらは抽象化されているので，オブジェクト生成してストリームをかぶせ，I/Oメソッドで制御すれば簡単にデータ通信ができてしまう。接続型(TCP)であるSocketクラスの使用例は，前回詳しく解説したので，前号で確認しよう。

また，サーバ専用のjava.net.ServerSocketというクラスも非常に使える。こちらはaccept()というメソッドで，クライアント接続の監視を行う。このメソッドは接続がくるまでずっと待機するので，簡単にサーバプロセスが作れる。

簡単な例を挙げておく。accept()の戻り値が，接続されたSocketオブジェクトなので，このままストリームに繋ぐことで，個別ユーザーに対応させるプログラム作成も簡単だ。

●1-5 Java特有の事情

このように，かなり便利なクラスが用意されているJavaだが，ソケット通信が確立するには，それぞれJavaVMが動いている環境にTCP/IPのインフラおよびプロトコルドライバが機能しているということが前提条件となっている。この辺注意が必要だ。

平たくいうと，モデム、ドライバ，プロバイダなどで，普通にTCP/IPでインターネット(LANでもいいが)ができる環境にしとかないと駄目ということ。なんにもネット環境が入ってないマシンで動かしても，Javaはその下回りを使って動くので，単体ではなにもできない。

これはJavaがさまざまなプラットフォーム上で仮想マシン(JavaVM：Java Virtual Machine)として動くことが多いという，Java特有の仕様と現状が関係している。

ほかの接続の形態として，JavaではRMI(Remote Method Innvocation)という選択肢もある。これは接続先のオブジェクトのメソッドを，直接実行できるというもの。ネットワーク全体を，自分自身の実行環境のように使用することが可能になる。

また，直接実行可能という点から，自分でプロトコルを作成するという，面倒なことをしなくて済むという利点がある。

だが，今回作成したゲームでは，このRMIを採用することができなかった。なぜかというと，実は大手ブラウザのバージョンによってはJavaVMの互換性の関係で，まったく使用できなかったりするからだ(Java Plug-inがインストールされていれば別だが)。

最近のバージョンでも結構まちまちなので，プレイしていただくユーザーの間口を広げるためには，どうしても手作業でプロトコルを作る羽目になってしまった。

なお，サーバ側の処理を作成する場合，Servlet(サーブレット)というJava製のサーバプログラムもある。これはサーバ側で動くJavaアプレットで，セキュリティが普通のJavaアプレット並みに強固なのと，ひとつのプロセス上で動作する点から，いちいちインタプリタが起動するPerlのCGIよりも動作が軽いという特徴がある(IISのISAPIよりはさすがに遅いし，PerlにもPerlisという高速化方法があるにはあるんだが)。

ほかにもいろいろあるが，Javaの場合，多くは便利なクラス，メソッドで覆い隠されているため，厳密に勉強しなくても，基本概念だけでプログラムは作れてしまったりする。

ネットワーク関係の基本的な知識や理論は，おそらくほかのライター陣が，私よりも遥かに詳しく説明してくれると思うので，実際にJavaでネットワークゲームを作る場合の取捨選択へ移ることにしよう。

■第2章：方式の選択

では，今回のテーマ「ネットワークゲーム」をJavaで作成する場合，Javaアプレット，Javaアプリケーションも含め，どの方式を選択するのがいちばんいいか，特性を挙げて考慮してみることにしよう。

●2-1 アプレットかアプリケーションか

さて，まずはJavaアプレットとJavaアプリケーション，どちらの形式で作成するかだが，両方とも一長一短がある。

ゲームを作成するうえでの長所，短所をまとめると，次のようになる(JDK1.1の場合)。

[Javaアプレット]
長所：①auファイルでサウンドを鳴らせる
　　　②一般的なブラウザがあれば実行可能
　　　③jarでclassファイルをまとめられる
短所：①セキュリティの制約が大きく，それに関係するアプリは面倒

[Javaアプリケーション]
長所：①セキュリティの制約が少ない
短所：①JDKかJREが必要
　　　②サウンドが鳴らせない
　　　③ファイルが山ほど生成される

まずはサウンドから。Javaが扱えるサウンド形式のau形式。元々Sunが提唱していて，UNIXユーザーにはお馴染みだが，これはJavaアプレットでのみ使用可能である。

ただし，サンプリングレートは低めの8kHz＆モノラルなため，正直いって結構しょぼい。最近のゲームミュージックのクオリティを考えると，ふた昔前といわれてもしかたないが，Javaの仕様なのでプログラマからは選択の余地なしって状態だ。だが，ゲームの効果音はないよか遙かにマシだろう。

Javaアプリケーションでは，これが一切使えない。まあ，サウンドを必要としないRogue系のゲームとかならいいかもしれないが。

次に実行環境。X68000ひと筋のユーザーにはあまり関係なさそうな話で申しわけないが，これだけインターネットが普及した状況で，もっとも簡単にJavaアプレットが動かせるという利点を使わない手はない。

これがJavaアプリケーションの場合，動かすには，必ずランタイムが必要となる。JDKに付属する"java"がそれに当たる。JREの場合はネイティブのコンパイラを使用して，キッカーを作らねばならず，面倒かつ環境依存がやや大きくなってしまう。

Javaアプリケーションの場合，ローカルで主にコマンド入力で実行することになるが，jarで圧縮したものをそのまま実行することはできない。これはJDKでもJREでも一緒である。

続いてjar機能だが，これはzip相当の圧縮解凍機能で，クラスファイルを1ファイルにまとめられる。

Javaはコンパイルすると，クラス単位で*.classファイルが生成される。そのため，プログラムが大規模になれば，classファイルが山のように発生してしまうのである。某社のJavaメーラーをお持ちなら，インストール先のフォルダを見てみ

よう。大量にclassファイルがあるはずだ。これは見た目およびHDエリアの容量の関係で，少々いただけないと思ってる。

また，ネット上からダウンロードして実行というJavaの特性上，ファイルがたくさんあるようではよけいな時間がかかる。圧縮してまとめられれば，ダウン終了までの時間を短縮できるというメリットが出てくるのである。

最後にセキュリティの問題。Java自身の強固なセキュリティの関係で，アプレットで作ろうとすると，そのままでは許可されない機能がいくつかある。詳細は前回解説したので省くが，通常，ネットワークゲームで必要なのは，ファイルアクセスとネットワークアクセスだろう。

まず，ファイルへは読み込みも書き込みも一切できない。ネットワークへは，自分自身がダウンロードされたサイトとしかアクセス，通信できないようになっている。

CLASSPATH上で実行する場合には，制限がなくなったりするなどの例外もあるが，基本的にJavaアプレットはサーバからダウンロードして遊んでもらわなくてはならないので，今回のような例では都合が悪い。

逆にJavaアプリケーションはセキュリティの制約がないので，スタンドアロンで動くものでも，ネット通信するものでも，なんの問題もなく作れる。

以上で2者の違いがはっきりしたわけだが，ネットワークゲームを作る場合，どっちに転んでもなんらかの問題が出てしまう。

だが，逃げ道がちゃんと用意されているので心配ない。

考えられる選択肢は次の2つ。

- **署名つきアプレット**
- **C/S方式で分担**

前者は「署名」をアプレットにつけることにより，ActiveXコンポーネントのように，実行先のユーザーに対して認証をさせる。

これは平たくいうと，「ダウンロードするJavaアプレットが危険なものかどうかの署名があるので，ユーザーご自身が危険かどうか判断してください」という感じだ。

署名情報はそうそう改竄されるものではないが，最終的にはユーザーの判断に任されることになる。

これだとすべての制限が外れるが，制限されていた機能を最初に使う瞬間，許可申請のダイアログが表示されてしまう（前号の署名つきアプレットの章を参照）。

このダイアログには，これを二度と表示しないというラジオボタンもあるが，セキュリティのかかった機能別に，いちいち出す仕組みになってしまっている。ファイルをアクセスするときに確認ダイアログ，ネットワークアクセスするときに確認ダイアログ，といった具合では，うざいうえにゲームを中断させられてしまう可能性がある。

次に後者のC/S方式。Javaアプレットが，ダウンロードされたサイトとのみ自由に通信できるということを生かして，ダウンロード元のサーバにJavaアプリケーションを設置し，それと通信，必要なデータの保存はすべてサーバ側に任せるという技だ。

別にサーバ側はJavaでなくてもかまいはしないが，Javaアプリケーションなら，なにも制限がないし，サーバアプリも作りやすいと思う。

これらの理由から，今回はC/S方式で，サーバ側をJavaアプリケーション，クライアント側をJavaアプレットとした。あえてJavaにこだわって，サーバもPure Javaで作成してある。

●2-2 プロトコル

続いては接続プロトコル。これは最初に解説したTCPとUDPの2つ。

TCPはなんといっても回線の信頼性が高いのがメリット。

UDPは単純で高速な転送向きだが，パケット消失の再送処理が面倒になる。

ゲームの内容によってはUDPで事足りる場合もあるが，信頼性を持たせるという利点のため，今回はTCPを採用する。

●2-3 Javaのバージョン

現在，大きく分けて3つのバージョンが広まっている。

- JDK1.0.2
- JDK1.1.x
- JDK1.2（Java 2）

JDK1.0.2はもう推奨されないため，基本的に使わないほうがいいのだが，これに対応しているJavaVM同士の互換性は，実はもっとも高い（ただしバグも多いが）。

JDK1.1.xは大手ブラウザなどの対応でかなり普及しているが，1.1.1と最新の1.1.7B，および各ブラウザ搭載のJavaVMバージョンの違いが大きく，特定のことをやろうとする場合の互換性が大変になる。

どう大変かというと，バージョンや動作プラットフォームをある程度限定し，推奨されないメソッドや追加・改変を考慮して作成しないと，メソッドが抹消されて動かなかったり，JavaVMの互換性がなくて動かなかったりする。たいていの場合，原因はブラウザ側にあるのだが，これで頭を悩ます技術者も多いと聞く（私もそうなのだが）。

JDK1.2は，この雑誌が店頭に出回る頃には徐々に浸透し始めてるとは思う。できればこれを使うのが，時事ネタとしていちばんいいのだろうが，いかんせん現時点ではブラウザのVMが対応していない（Java 2のJava Plug-inもリリースされているが，JDK1.2に付属のみという形になっている）。

今回の本誌のテーマは「ネットワークゲーム」なんで，できるだけ多くの人に遊べる環境を提供したい。でないと，再配布禁止のJDKをわざわざダウンロードしてもらうような形になってしまう。ちなみに最新版のJDK1.1.7B（Windows版）で約9Mバイトもある。遅めの通信速度でネットしている場合などには，ダウンロードの料金も馬鹿にならない。

やはり，一般的に普及しているブラウザで動かしてもらうのがいちばんということなんだろうて（だが，この選択をしたばっかりに，あとで製品バージョンでの動作チェック地獄が待っていたのであったが……）。

では次に，Javaでゲームを作成する場合のワンポイントテクニックをば。

■第3章：Javaゲーム作成テクニック

まず，ゲームというものは一般的に，画面の見た目が非常に大事だ。アニメーションしたり，ガリガリ動くならなおさら。

この辺り，異論が多々あるとは思うが，画面が綺麗なだけで内容が伴わなけりゃ，単にクソゲーとなってしまうことぐらい，私だって十二分に理解している。が，それでも見た目に気を使う必要があるのはいうまでもない。

で，Javaはゲーム作成にも向いているのだが，実はそのまま作ってしまうと画面表示系において，多少見栄えが悪くなってしまう欠点がある。

具体的にいうと，それは「ちらつき」と呼ばれるもの。ゲームをよくやる方にはいまさらだが，少々昔のアクション系ゲームなどで，キャラクターや背景がちらちらとして，はっきり表示されないという現象を見たことがないだろうか？　これは見づらいうえに，印象が非常に悪いのだが，この現象が出てしまう。

このちらつきも含め，Javaでゲームを作る際によく使用されるテクニックを，いくつか紹介しておこう。アニメーションなど，頻繁に書き換えを伴うゲームの場合，これをやるのとやらないのでは差が出てしまうので，とりあえず知っていたほうがいい。もちろんゲームに限らず，グラフィックを頻繁に書き換えるようなプログラム全般で効果がある。

これらはJava発生当初から使われている技法なので，もうお馴染みすぎるかもしれないが，ご勘弁を。

●3-1 アプレットのループ構造

さて，ゲームといえばループ構造がお約束。たいていの場合，終了するまでひたすら入力・内部計算・画面書き換えの繰り返しでゲーム処理を行うのが基本だろう。

Javaアプレットではこれらに相当する構造と，便利なメソッドがすでに用意されている。

これらが，Javaアプレットのメソッドの中で，ゲーム作成時に直接関係のあるものだ。

paint()	全体の描画
update()	画面クリアとpaint()呼び出し

repaint()　update()呼び出し
run()　ループ
stop()　ビューアの停止時等

Javaアプレットは，start()メソッドからrun()が実行されてループが始まる。画面が隠れたあとの再描画は，paint()が呼ばれるような仕組みになっている。

run()が終了するということは，アプレットが終了すると同じことである。実際の事後処理はstop()メソッドに書くとしても，ここにメインの無限ループを設置するのが普通だろう。

実際にゲームが走ると，ループ，入力イベント，その他諸々でゲームの処理が発生し，画面の書き換えが必要となる。プログラム側で書き換えが必要になったときにrepaint()を呼ぶと，そこからupdate()が呼ばれ，さらにpaint()に行き着く。

run()のループ構造と，イベント処理時に適宜repaint()を呼ぶことによって，画面書き換えが行えるというわけだ。

●3-2 update()のオーバーライド

java.applet.Appletクラスのupdate()メソッドは，次のようになっている。

update()xメソッド

```
public void update(Graphics g) {
 g.setColor(getBackground());
 g.fillRect(0, 0, width, height);
 g.setColor(getForeground());
 paint(g);
}
```

これらのメソッドは順に，

- バックグラウンドカラーに設定
- 最大領域まで塗りつぶし
- フォアグラウンドカラーに設定
- 全描画メソッド呼び出し

という処理をしている。

引数のGraphicsは，グラフィックコンテキストの抽象クラスだ。Javaで画面への書き込みを行うには，基本的にこのクラスを使うと思っていい。

このクラスには矩形や円だけでなく，イメージパターンや3Dポリゴンを表示する機能もあるので，スプライトのようなゲームも3D表示もお任せって感じだ。

話をupdate()に戻そう。

つまりこのままだと，全描画を行うpaint()メソッドを呼ぶ前に，FillRect()メソッドで，全エリアをまっさらに塗りつぶしてしまうのだ。

Javaでアニメーションアプレットを作るときに発生する，ちらつきの原因のひとつはこれだ。書き換えのたびに全画面をいったん塗りつぶすから，同期が取れずにちらちらとノイズが走るように見えるっていうわけだ。

いちばん簡単な解決方法は，これをオーバーライドして，塗りつぶしをしないようにすることだ。単に塗りつぶしを削除するだけだったりするのだが，こうするだけでも結構変わってくる。

オーバーライド後のupdate ()

```
public void update (Graphics g) {
 paint (g);
}
```

●3-3 ダブルバッファリング

次に，paint(Graphics g)メソッド。

普通に描画する場合，これをオーバーライドして，引数のGraphicsオブジェクトの描画メソッドなどで書き換えを行うが，描画が複雑になってくるとこれもちらつく原因になる。いちいち書きながら再表示を行っているからなんだが。

だいたい，普通はこんな感じで，直接引数のGraphicsオブジェクトのメソッドを使用して描いているだろう。

簡単なpaint()例

```
public void paint(Graphics g) {
 // 矩形を表示
 g.setColor(Color.red);
 g.drawRect(0, 0, 32, 32);
 g.setColor(Color.blue);
 g.fillRect(64, 64, 32, 32);
}
```

そこで，画面からは見えない書き換え専用エリアを用意しておき，画面に表示するまではそちらにガリガリと描画，終わったら一気に画面へ高速転送という方法を使う。

イメージパターンをあらかじめ作り上げて高速転送という，ゲーム作成上でよくある技法のJava版だ。しかも，テクニックとはいえ非常に簡単で，描画するためのGraphicsオブジェクト，オフスクリーンをもうひとつ用意し，そちらに書いておくだけである。

先ほどのソースを，このテクニックで書き直すとこんな感じになる。

```
import java.awt.*;

Image off_i;       // オフイメージ
Graphics off_g;    // オフスクリーン

public void init() {
 // オフスクリーンを初期化
 off_i = createImage(width, height);
 off_g = off_i.getGraphics();
}

public void update(Graphics g) {
 // paintを呼ぶのみ
 paint(g);
}

public void paint(Graphics g) {
 // オフスクリーンに矩形を表示
 off_g.setColor(Color.red);
 off_g.drawRect(0, 0, 32, 32);
 off_g.setColor(Color.blue);
 off_g.fillRect(64, 64, 32, 32);
 // 一気に転送
 g.drawImage(off_i, 0, 0, this);
}
```

こうすれば，書き換えと表示が完全に同期し，スムーズな描画が行われる。

●3-4 描画領域のクリッピング

最後に描画領域の指定。

paint()メソッドではすべての領域を書き直すため，必要のない場所までもう一度書き直すことになる。

そこで，書き換える矩形領域を指定するという方法がある。Graphicsオブジェクトのclip Rect()メソッドを使う場合と，repaint()メソッドを使う場合の2パターンがある。どちらでも構わない。

指定領域クリッピングの例：
　(0，0〜32，32の指定の場合)

```
g.clipRect(0, 0, 32, 32);
repaint(0, 0, 32, 32);
```

前者はpaint()メソッドの内部に書き，後者は各処理から書き換え指示に使うのが，普通の使い方だ。

こうすると，描画される部分は，引数で指定したエリアのみになるため，部分的な更新で構わないとわかっている場合に，高速化することが可能になる。

特に，今回作成したゲームでは，マトリクス状のフィールドマップがメインであるため，セルの部分書き換えが頻繁に行われる。このような場合は，特に有効だ。

●3-5 入力関係

ゲームといえばデバイスからの入力も大事だ。一般的にPC上で動くことが多いJavaのデバイスは，マウスかキーボードになるだろう。

JDK1.1.xにはリアルタイムのキーボード，およびマウスイベントのハンドラが用意されている。

キーボードハンドラ I/F
　java.awt.event.KeyListener
マウスハンドラ I/F
　java.awt.event.MouseListener

これらは「インタフェイス」と呼ばれるJavaの抽象化機能であり，多重継承ができないJavaで，ほかのクラスの抽象関数，定数を継承できるという，Javaプログラマは勉強必須の仕様だ。

インタフェイスに記述される関数は，すべてpublicでabstractになる。つまり，関数の型のみで，実体はインプリメント先のクラスでオーバ

ーライドしなければならない。

変数の場合は，必ずstaticかつfinalになる。したがって#defineのような定数として使用することになる。

インタフェイスをクラスに追加するには，次のようにimplementsキーワードを使う。

今回のI/Fの場合の例：

```
import java.awt.event.*;

class Smaple extends Applet implemen
ts KeyListener,MouseListener{

 // 抽象処理をオーバーライド
 public void KeyTyped(KeyEvent e){
 .
 .
 .

}
```

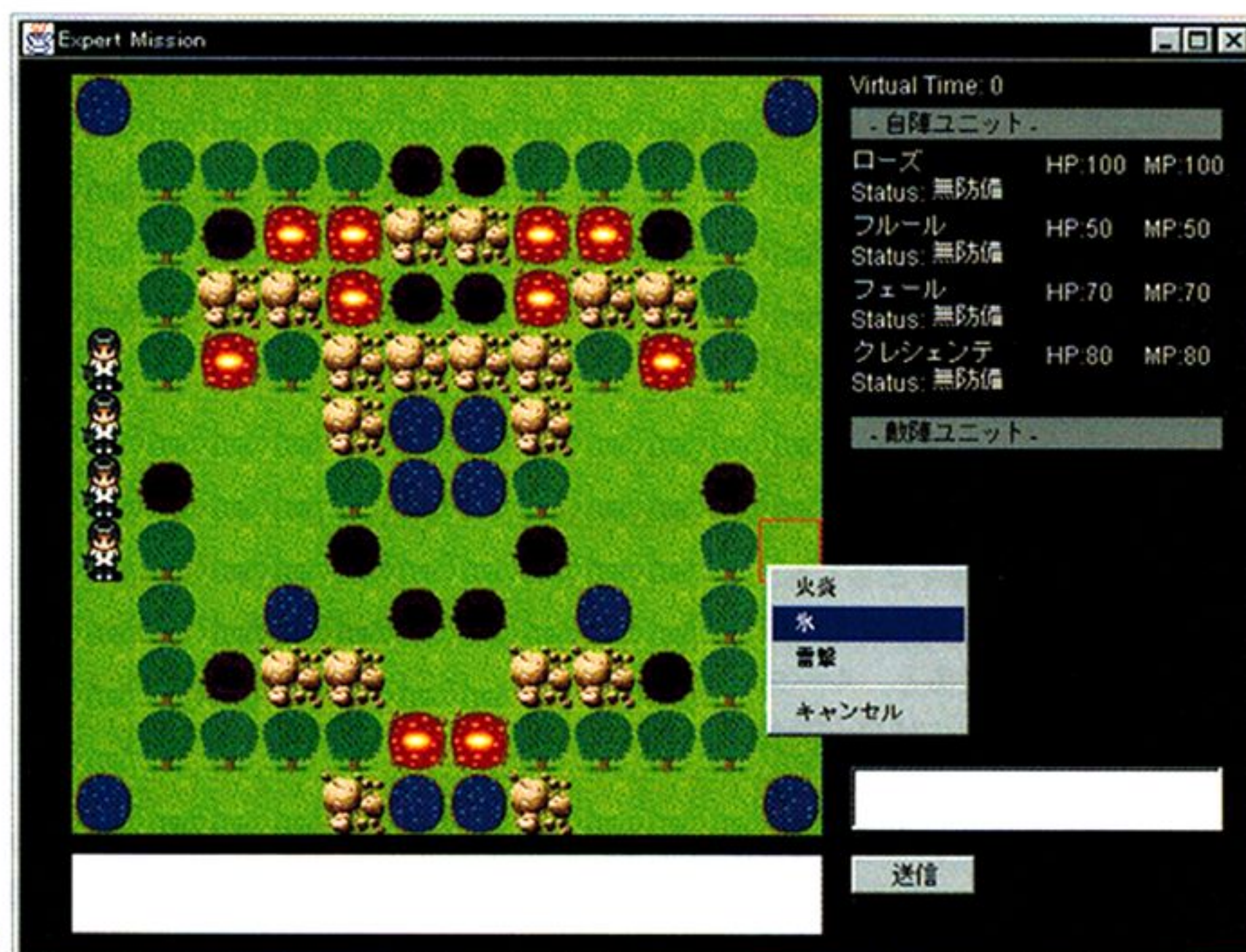
これらがキーボードとマウスのインタフェイスに入っているメソッドだ。

```
KeyListener メソッド
 void keyTyped (KeyEvent e)
 void keyPressed (KeyEvent e)
 void keyReleased (KeyEvent e)

MouseListener メソッド
 void mouseClicked (MouseEvent e)
 void mouseEntered (MouseEvent e)
 void mouseExited (MouseEvent e)
 void mousePressed (MouseEvent e)
 void mouseReleased (MouseEvent e)
```

図　ゲーム画面

クラス宣言の例のようにimplementsしたら，すべてのメソッドをクラス内に記述し，イベントの内容に従って処理を書くだけだ(抽象メソッドは継承先のクラスに必ず記述しなくてはならない)。

引数のクラスに，入力キーの内容，マウス座標などのデータがメンバ変数として入っているので，あとはプログラムの内容に応じて処理をするのみとなる。

サンプルは用意していないが，添付したゲームソースの，MainGame.java内で同様の処理を行っている。

■第４章：内部解説

ここから，今回作成したゲームの要所を解説しよう。このJavaゲーム「Expert Mission」は，C/S方式で動くPure Java 100%のタクティクスシミュレーションである。

本誌に添付したソースコードは，読者の皆さんがコンパイルして，ローカルで実行しても，セキ

COLUMN　努力の足跡

まずはJavaで，２号テーマのネットワークゲームを作るということから始まった。

回線の最低速度がどれぐらいになるかもわからない，実行環境の最低スペックがどうなるかもわからない。おまけにもともと遅い(失礼)Javaである。

最近はフリーなJavaのソースも出回っている。JDK1.2のソースも公開された。どうせならX68000で動くようにするという手もあったのだが，動くようにしたとして，はたして使いものになるかどうか保証がない。実行はできても文字表示の書き換えが目で見えるよーなほど遅かったらどーしよ，ってのがある。

いうまでもなく，ドノーマル10MHzでは話にならないはず。060専用とか，零式専用になるにしても，べんちマシンですらあの調子なので高速動作は望めない。

Javaの通信パフォーマンスが見えない状況でいろいろ話しあっても，すべて机上の空論でしかないということになり，私はとある環境を使ってJavaのソケットパフォーマンスを調べてみることにした。

実験に使ったマシン，環境のスペックはだいたいこんなカンジ。

[サーバ側]
MMX Pentium 266MHz/128M/WinNT4.0 sp3

[クライアント側]
Pentium 166MHz/64M/Windows98 final β

[回線状況]
LAN 128k ただし混雑時のため，ややパフォーマンス悪し

[Java環境]
開発環境と同じ

[テストプログラム概要]
サーバ/クライアントともに，送受信以外の負荷はほとんどないプログラム（TCP単体接続・データ垂れ流し処理のみ）

C/Sは同一セグメント，同一Hubへのコネクト(テスト時間が日中のため，実質80,000～120,000byte/sec程度の転送率)

で，テストを行った結果は，このとおり。30回近く行って，平均を取ってある。

<テスト１>
32ビットデータ0(int型)送信

一方向通信：
600～900 byte/sec[150～220 sent/sec]

一方向×2(スレッド)通信：
200～400 byte/sec[50～100 I/O /sec]

<テスト２>
8ビットデータ (byte型) 送信

一方向通信：
200～500 byte/sec

一方向×2(スレッド)通信：
100～200byte/sec[40～80 I/O /sec]

最初は目を疑ったが，何度やってもこれ以上のパフォーマンスは得られなかった。

しかも，スレッドでもうひとつストリームを繋ぐと，パフォーマンスが一気に下がる……。

最初はキャラがガリガリ動いて，相手のキー入力も受け取るゲームを想定していたのだが，このスペックでこれでは，ちょっと期待しているものが作れない可能性がある。

インターネット上で通信対戦できるアプレットは何度か見たことがあったが，ボードゲームがほとんどだった。アクションがあると思ってアクセスすると，予定とか未定で誤魔化されたことが何度かあったし……。

市販のネット通信ゲームではどーやってインフラの違いすぎる穴を埋めているのだろう。ある程度の推奨スペックを明記すればいろいろやり方もあるだろうが，Javaでやるとなると，強固な仕様が邪魔をしてくれてカリカリな微調整はできそうにない。

やっぱりいろいろ壁があるんだろう……最初は多人数同時プレイに挑戦しようという案まであった。

何度かゲーム案を考えては没にするという手順を踏んだあげく，今回のようなタクティクス対戦シミュレーションになったのだ。

ュリティの関係上サーバにつないで遊ぶことはできない。あくまでテクニック参照用としての資料という形式をとることになってしまった。

ゲームを遊ぶには，用意したゲームサーバにアクセスし，そこからダウンロードされたアプレットで動かしてもらうことになるので，どうかご了承いただきたい。

これもJavaの特徴であり，専用サーバを設置するという形になった以上，しかたないのだが，今後最新版のソース公開や，その他の対応は「Expert Mission」公式HPでも行っていく予定になっている。

ゲームだけがしたい方，ソースに興味のある方に限らず，一度はお立ち寄りいただけるとありがたい。

COLUMN 今回の内容に関連するリンク先

「Expert Mission」Official HP
http://www.antai.suginami.tokyo.jp/em/

JDK1.1
http://java.sun.com/products/jdk/1.1/index.html

JDK1.2（日本語）
http://java.sun.com/products/jdk/1.2/ja/

Java Plug-in 1.1.1
http://java.Sun.COM/products/plugin/1.1.1/index-1.1.1.html

Java Plug-in 1.2（日本語）
http://java.Sun.COM/products/plugin/index_ja.html

Java Source Code
http://java.sun.com/products/jdk/1.1/source.html

IBM alphaWorks HP（Javaコンパイラ"Jlikes"）
http://www.alphaworks.ibm.com/formula/JikesOS/

Symantec Japan
http://itools.symantec.co.jp/

Symantec Japan 製品情報（Visual Cafe等）
http://itools.symantec.co.jp/product/index.html

Symantec JITspeed（Win, Mac）
http://itools.symantec.co.jp/product/jitspeed/jitspeed.html

●4-1 サーバ側内部構造

サーバのソースは，今回は諸般の事情により公開を差し控えさせていただいた。現在も急ピッチで更新が続いている。この本の発売にはサーバ上でちゃんとしたものを公開する予定だ。それ以降も随時改変されることだろう。

このJavaサーバは，ざっと以下の仕事をこなしている。こちらは先ほど書いたとおり，Javaアプリケーションで動作している。

・プレイヤーアクセス待ち
・登録プレイヤー管理
・プレイヤー割り振り
・ゲーム進行
・データ集計

アクセス待ちには，ソケットの項で解説したServerSocketクラスを使用し，ユーザーの割り振りを行っている。

今回は時間の関係で，対戦相手の指定ができない構造になってしまった。詰め込み方式でアクセスした人と対戦させるので，誰に当たるのかわからない。このあたりはそのうち改善されるかもしれない。

今回は，ゲームを進めるうえで，専用のプロトコルを作成した。その他，もっと詳しい内部構造についても後日まとめることになるだろう。

●4-2 クライアント側

クライアントはJavaアプレットで作成してある。ブラウザでも動作するが，ゲーム自体はフレームで別ウィンドウ表示になるため，クラス構成は次のようになっている。

ExpertMission.java	本体＆ログオン
MainGame.java	メイン
sSprite.java	スプライト
UnitChara.java	ユニットキャラ
Field.java	マップ管理
StreamManager.java	通信
ImageStocker.java	イメージ管理

これ以外にインタフェイスとして，

SpriteNums.java	スプライト
UnitParams.java	ユニット共通
Protcols.java	プロトコル

を作成している。

●4-3 VirtualTime

このゲームは擬似的なリアルタイムのシミュレーションである。ゲーム全体に流れる時間は実際の時間ではなく，サーバ側でカウントされる擬似時間を採用している。

これを「Virtual Time」と（勝手に）呼んでいる。サーバ側から定期的にクライアントへ通知する仕組みにした。

クライアントは，このカウントをサーバから受け取り，イベント動作開始などの処理を行う。

サーバ側から通知する仕組みにした理由としては，同期性の問題がある。プレイヤー同士のプラットフォーム性能が違うのはもちろんのこと，通信回線その他の理由による多体性のずれを考慮してのことだ。

■5章：ゲーム解説

今回制作したゲームはタクティクスタイプのシミュレーションゲームである。

遊ぶためには，専用のゲームサーバが必要な形式になったのだが，これはソフトバンクパブリッシングが提供および設置していただくことになった。

ゲーム自体の世界観，キャラデザイン，CGは野沢絵美嬢に担当していただいた。ご協力感謝致します>野沢嬢

また「Expert Mission」の公式HPを，ライターのぺけ氏に設置していただいた。サーバ提供感謝致します>ぺけ氏

うーむ，感謝することばかりになってしまったが，ゲームの内容に関する解説は野沢嬢にまとめてもらったのでそちらを参照してほしい。

■6章：Javaの変遷と現状

最後に，前回紹介した，Javaを取り巻く状況についての変更点をいくつか補足しておくことにする（どうせ本が出る頃には変わってるんだろうけど）。

●ブラウザのJavaVM対応状況について

Netscapeの日本語版は4.5と4.07がリリースされた。JavaVMの対応状況は細かいバージョンによってまちまちだが，最新版ではRMIにも対応するようになった。本体だけでのJDK1.2の対応はまだまだ未定というのが現状である。

マイクロソフトとサンマイクロシステムズの裁判はいったんサン側の勝利というかたちになった。マイクロソフトはサン互換のJNI対応VMを差し替えという形で配布することになった。マイクロソフトのサイトでダウンロードすることも可能だ。

●サンマイクロシステムズ本家のリリース状況

JDK1.1.xの最新版は，1.1.7Bになっている。1.1.6からJITコンパイラが搭載され，処理が高速化された。

JDK1.2はようやく正式版（Java 2）がリリースされた。セキュリティ面がより強固になっているほか，変更点が書ききれないほど多数ある。

また，条件つきではあるが，Java 2のソース公開という思い切った手に出ている。製品にする場合には有料なものの，参照だけなら無料だ。

余談だが，IBMのalphaWorksからもJavaコンパイラ「Jikes」のソースコードを無償公開している。

C，C++などで記述されているので，コンパイラ技術に興味のある方はダウンロードするのもいいだろう。

さまざまな問題のせいで，最近Java熱が冷めてきたように感じていたのだが，オープンソースという思い切った手で，また最近の流れに乗ったともいえる。これでどうなっていくのか楽しみなところだ。

なお，今回紹介した製品も含め，関連するサイトのURLはリンクリストにまとめておいた。COLUMNを参照してほしい。

特集 ネットワークゲームの地平へ

Java対応対戦型ゲーム Expert Mission ユーザーズマニュアル

野沢絵美 *Nozawa Emi*

Javaプログラムで作られたネットワーク対戦専用の疑似リアルタイム制のタクティクスシミュレーションゲーム〝Expert Mission〞。今回はExpert Missionの遊び方から世界観などの説明をします。Javaでネットワーク対戦ゲームを作りたい方は，プログラミングやコンセプト作成の参考にしてください。

●ゲームの舞台設定●

Expert Missionの舞台となるのは異世界にあるミッションスクールです。このミッションスクールは，将来，神官の地位を得て各地の教会を守護し神様の教えを伝える女神官となる優等生が集まり教育を受ける全寮制スクールです。学校への入学希望者は8歳から教育を受けることができます。教育費などは教会から出るので基本的には無料です。たいがいは寄付金としていくらかの寄附金を納めるものですが，寄付金の額にかかわらず誰もが平等に教育を受けることができます。能力があれば身分を問わずなれる神官職は人気のある職業ともいえるでしょう。

この世界での教会は説法，学校のない地域での子供向け初等教育，傷病人の治療，薬の調合，農業指導といったさまざまなことをこなす万能施設なのです。その神官ともなると，学問，医学，薬学，農業に詳しくなくてはなりません。そして，世の中の情勢が不安定なこの世界で，食料と資金が豊富な教会は略奪の対象になりやすいので，自分を守る術を身につけておく必要があります。

そう，神官たるもの文武両方に精通していないとダメなのです。学校では宗教の教えはもちろんのこと授業の一環として武術，魔法，罠の設置などを教えています。武術の授業では，学校の敷地内で実際に戦闘もします。実戦の授業でユニークなのが戦闘時の服装。学校には制服があるのですが，このときはどんな服装をしてもOKなのです。そう，いつ何時，どんな場所，どんな格好でも戦えるのが理想のため，なにを着てもOKなのです。となると年頃の女の子のこと，みんな結構気合を入れておしゃれをしちゃうのです。だってめったにおしゃれなんてできないミッション系スクールです。こんなところで女の子がでちゃうってものなのです。戦闘はそっちのけでファッションを争うやからもいるし，素敵なおねーさまにはファンクラブなんかもできちゃってます。

Expert Missionはそんな着飾った女学生になって戦う，対戦ゲームです。

●Expert Missionキャラクター紹介●

Expert Missionの戦闘に参加する女の子は4人。キャラクターによっては力がある子や，魔法が得意な子，頭で罠を回避する子と1人ひとりできることが異なります。また，キャラによってHP，MP，力，防御，スピード，移動距離が違います。女の子の特徴を生かして戦闘を進めます。

① 戦闘タイプ

武器で相手を倒す直接戦闘型です。移動距離は少ないですが攻撃・防御に優れています。魔法や回避の能力はありません。

名前	ローズ
年齢	15歳
トレードマーク	ピンクのリボン
武器	スピア
アビリティ	攻撃・防御・移動

キャラ設定のこだわり：

行動系なので洋服は動きやすいボディコン系ミニスカと，それだけじゃあシルエットが寂しいのでリボンをあしらったデザインにしてみました。行動的で明るい性格の子なので色も白とピンクを中心に。自然と彼女の趣味はリボン集めに決定。

攻撃がメインで主戦力になるキャラ。ワナとよく運動させて戦わせてあげよう。

② 魔法使いタイプ

魔法を使って相手を倒します。魔法は四方に影響しますので，複数人数を一度に攻撃することも可能です。ある程度の直接攻撃もできますが，力と防御力はそんなにありません。

名前	フルール
年齢	16歳
トレードマーク	花
武器	ロッド
アビリティ	回避・攻撃・防御・移動

キャラ設定のこだわり：

ギャルゲーには必ずメガネっ娘は必要とのことで，登場したメガネ魔法使いです。洋服は花をあしらった一見お嬢様風のフリルのワンピースですが，腕の部分に透ける素材を取り入れた大胆なデザインとなってます。見た目はおとなしそうですが，行動するととんでもない過激な本性が現れるという性格を服に反映させてみました。

③ 頭脳タイプ

罠を仕掛けて相手を倒します。仕掛けられた罠の解除もできます。直接戦闘や魔法攻撃はできません。

名前	フェール
年齢	17歳
トレードマーク	アタッシュケース
アビリティ	罠設置・罠解除・罠発動・移動

キャラ設定のこだわり：

アタッシュケースにいろいろなツールを持っているので，自然とスーツ姿になっちゃいました。この世界に会社があるのか～？　ってつっこみはナシにしてください（はっきりいってこの世界における縫製の時代考証はめちゃくちゃです）。冷静沈着で自分の正体を隠す人が好み，美人にしか似合わない紫色をイメージカラーに使いました。

④ 忍者タイプ

隠密行動で相手を倒します。武器による戦闘と罠の回避が得意です。魔法は使えません。

名前	クレシェンテ
年齢	14歳
トレードマーク	三日月
武器	三日月短剣
アビリティ	回避・回避解除・攻撃・移動

キャラ設定のこだわり：

キャピキャピした若さを思いっきり見せつけるミニスカ，背中出し系のデザインです。胸のトップで服全体を支えられるのはCカップ以上の証拠。月のアクセサリーを身につけているし，髪はトップでお団子型に結んでいるしで，某月アニメーションの主人公に似ていますが偶然の一致です。カラーは蛍光系のピンクで決めてみました。若いのでハデなピンクも躊躇なく着れるのです。

＊設定裏話：実はExpert Missionの世界設定は「女の子が戦う」をコンセプトにミッション スクールとの舞台が生まれたのです。いいんです。自分の好きなことから世界は生まれるのです。だって，ミッションスクール系の女の子って興味あるでしょ？

走行環境

Expert Missionで遊ぶには次の環境が必要になります。あらかじめご用意ください。

クライアント走行環境

・インターネットに接続された，JRE 1.1.3相当以上が走行可能な，WWWブラウザなどのソフトウェアプラットフォーム。

・マウスなどのポインティングデバイス。
・640 × 480 ピクセル以上，
・256色以上でゲームを表示できるデバイス。

クライアント走行環境の例（x86系Windows ファミリー製品＊1）

●ハードウェア
マシン：Pentium/150MHz以上を実装したAT互換機
メモリ：32 MB 以上
ディスプレイ：640×480ピクセル以上，
256色以上を表示するビデオカード
●インターネット接続
Winsock 1.1 以上
Expert Missionを無理なく運用するために，通信速度は28.8kbps以上を推奨
●ソフトウェアプラットフォーム
Java Activator
（Sun Microsystens［Java Plug-in］）推奨
Microsoft Internet Explorer 4.01 SP1
（Microsoft Corporations）以上
Netscape Navigator 4.06
（Netscape Communications）以上
HotJava 1.1.4（Sun Microsystems, inc.）以上＊2

＊1 ここで言及している「x86系Windowsファミリー製品」とは，Windows 95（OSR2以降），Windows 98，Windows NT 4.0（X86）のことです。1998年12月15日公開されたJava Native Interface（JNI）をサポートした Java仮想マシン（Java VM）導入済みであることを前提としています。この件についての詳細はマイクロソフトのMicrosoft Internet Explorerページの該当項目を参照ください。
＊2 HotJavaにはJREが必要です。詳細はSun Microsystems, Inc.のWebサイトを参照ください。

以上のプラットフォーム以外に，ベンダー各社から出荷されているJDK 1.1.3，JRE 1.1.3以上のJava VMを持つ開発環境などに付属のアプレットビューアでも走行。アプレットビューア については各社の説明書を参照ください。

確認済みクライアント環境（1999年1月現在）

SONY "VAIO"（PCG-505EX/64）
CPU：MMX Pentium/233MHz，
メモリ：64MB，
ビデオチップ：NeoMagic MagicGraph 128ZV+
マウス：USB マウス
Windows 95 OSR 2.1 with USB Support

「走行環境の例」以外のExpert Missionの走行環境

「Java」なので，Windowsシリーズ以外にも，Macintosh（ただしPower Macintosh以上），UNIXとUNIX互換OSなどで，Javaの要件を満たすものがあれば稼働するはずですが，保証はいたしません。まったく動作しない，途中で止まる，といった可能性があります。Java Plug-inをインストールすれば，動作する「可能性」はかなり高くなると推測します。ただし，Java Plug-in自体が Windows 95，Windows 98，Windows NT 4.0，そして，Solaris 2.5/2.6にのみ対応している点にも，またご理解を願います。Java Plug-inには，USバージョンとインターナショナルバージョンがあり，Expert Missionは後者を必要とします。Java Plug-inの推奨環境確認とダウンロードは，Sun Microsystems,Inc.のホームページにてお願いします。

■Expert Missionサーバ

ゲームをプレイするには，専用ゲームサーバに接続する必要があります。サーバはソフトバンクパブリッシングに設置してあります。

＊ゲーム中は常にインターネットに接続されていることが前提になります。プロバイダやゲーム用サーバの混雑状況によっては，接続できなかったり，エラーが発生したりする場合があります。あらかじめご了承ください。

●ゲームのやり方●

■概要と目的

Expert Missionはネットワーク対戦専用の疑似リアルタイム制のタクティクスシミュレーションゲームです。プレイするには必ず相手が必要ですが，サーバにアクセスしている人間同士で対戦を楽しめます。プレイヤーはユニットと呼ばれる，それぞれ特性の異なるキャラ（女の子）を，フィールド上で4 on 4で戦わせます。時間内に，相手のユニットを全滅させれば勝利したと見なされます。ゲーム上では制限時間がありますので，時間内に勝負がつかなかった場合は，残ったユニット数とHPで判断されます。プレイヤーの成績はサーバ上に常に記録され，ランキングが集計されます。

■ユーザー登録

ゲームを始める前にユーザー登録を済ませます。その場でIDとパスワードが発行されます。次からこのIDとパスワードを利用してゲームサーバにアクセスしてください。一度ユーザー登録を済ませましたら，次回からはユーザー登録の必要はありません。

注意！　IDとパスワードの再発行は行いません。表示されたIDとパスワードは必ず紙などに記録しておいてください。

Expert MissionサーバURL
http://xlabs.pub.softbank.co.jp/em/

■サーバにログオンする

ブラウザ，アプレットビューアなどで以下のサイトにアクセスしてください。IDとパスワードを入力し，サーバ上で対戦を待っている人がいればゲームが始まります。誰もアクセスしていない場合はウェイトメッセージが表示されます。そのまま待っていると次にアクセスした方と対戦できます。

ゲーム用サーバURL
http://xlabs.pub.softbank.co.jp/em/em.html

■画面構成

画面1がゲームのメインとなる画面です。これはブラウザ，アプレットビューアの内部に表示されます。

①フィールド
戦闘を行うフィールドマップです。
②自キャラ
自分が操作するキャラクターです。
③敵キャラ
ネットを通じて対戦する相手のキャラです。
④自ステータス
自分のキャラのステータスです。
⑤敵ステータス
相手のキャラのステータスです。
⑥メッセージエリア
各状況その他の指示が表示されます。

■フィールド（地形紹介）

戦闘の舞台となるフィールドマップは12×12セル固定です。マップはさまざまな種類があります。マップは次の要素で構成されます。

(1)平地：ごく普通の平地です

(2)岩場：動きにくいため，攻撃力，防御力が低下します

(3)沼地：これも動きにくく動きが低下します

(4)油：滑るので攻撃を受けると100%の確率で飛ばされます

(5)木：障害物です。通り抜けることができません

(6)火炎：油だまりに火がついたものです。上に乗るとダメージを受けます

各フィールド要素にはさまざまな隠された効果があります。いろいろと遊んで確かめてみましょう。

■操作方法

●フィールド上の操作

操作はキーボードから行います。

移　動：矢印キー 上↑ 下↓ 左← 右→
決　定：Zキー
キャンセル：Yキー

[通常時]

画面上のユニットは常にプレイヤーの指示を待っています。あなたが操作できるのは自陣のユニットのみです。まず，ここで行える動作はユニットの選択です。ユニットに指示を出すには，そのユニットがいる場所にカーソルボックスをあわせてZキーを押します。

[選択時]

自陣のユニットを選択すると，行動可能なアビリティがアビリティエリアに表示されます。現在選択中のユニットに指定可能なアビリティは通常表示されます。灰色になっているアビリティは指定できません。

[移動]

フィールド内の別のセルへ移動します。「木」「やられたキャラ」は通ることができません。「火炎」では通った瞬間および，VTごとにダメージを受けます。

移動先指定：目的地のセルまでの道筋をカーソルボックスで1セルずつ選択します。追加したくない場所や罠のありそうな場所を避けて移動することが可能です。

●攻撃

攻撃アビリティを選択すると，攻撃方向を決定するための矢印が，ユニットを中心に表示されます。攻撃範囲はユニットの種類によって違います(範囲がカーソルボックスで表示されます)。なお，直接攻撃の方向は4方向です。攻撃が成功すると，受けたキャラクターはランダムで1セル分飛ばされることがあります。フィールドの端にいれば飛ばされませんが，障害物に激突してダメージを受ける場合もあります(障害物を背にしているときも一緒です)。飛ばされた先にユニットがあれば，同様にダメージを受けます。

攻撃：攻撃方向の矢印を選択します。

[防御]

防御アビリティを選択すると，なにも行動しない代わりに，攻撃を受けたときのダメージが最小限になります。また，攻撃が成功しても飛ばされることはありません。ただし，防御を延々続けていると，いずれガードが崩されてしまいますので注意してください。

防御：防御のアビリティを選択します。

●魔法詠唱

魔法詠唱を選択すると，魔法の呪文を唱えます。詠唱から発動までは魔法の種類によって，いくつかのVTを必要とします。また，魔法詠唱にはMPを必要とします。

足りない場合，発動しなかったりおかしな現象が出ることがあります。魔法は唯一斜め方法を含む複数フィールドに攻撃を行えます。仲間を巻き込まないようにご注意ください。なお，魔法詠唱終了後は，まったくの無防備です。

魔法詠唱：唱えたい魔法の種類を選択します。

魔法の種類：魔法使いが唱えることのできる魔法は，次のような種類があります。

(1)火炎
(2)氷
(3)雷

●罠設置

罠設置を選択するとその場所にトラップを設置することができます。これもいくつか種類があり，地雷，タライなどで発動します。一度設置した罠は，敵・味方の区別なく影響を及ぼします。これも仲間を巻き込まないように注意してください。罠師もその場所からいったん移動すると，他ユニットと同様にトラップに引っかかります。

罠設置：設置したい罠の種類を選択します。

発動：キャラクターが罠を設置した場所を追加すると自動発動します。

罠の種類：罠師が設置できるトラップには，次のような種類があります。

(1)地雷
地雷を設置した場所を通過すると自動的に爆発します。大きなダメージを受けます。
(2)たらい
たらいの設置場所を通過するとたらいが落ちてきます。その場から動けなくなります。
(3)落とし穴
落とし穴を設置した場所を通過するとしばらくの間動けなくなります。

■時間の概念

ゲーム上では"Virtual Time"(以下VTと省略)と呼ばれる仮想的な時間が流れます。このVTがすべてのキャラクターの行動を制御することになります。このVTは999からカウントされ，0まで流れます(1VTは約1秒強です)。 各ユニットにはスピードの概念があり，行動終了後から次に行動できるまでの時間をこのVTで表します。たとえば，スピードが10のユニットの場合，行動終了後に10VT経過しないと次の行動がとれません。ユニットごとにVTの違いがあるので，行動が速いユニットや遅いユニットが存在します。あらかじめユニットの特徴を把握して作戦を考えてください。

■勝敗

相手のユニットを全滅させたほうが勝ちです。また，ゲーム上では制限時間がありますので，時間内に勝負がつかなかった場合は，残ったユニット数とHPが多いほうが勝ちになります。ゲーム中に相手が対戦を放棄したり，サーバからログオフした場合は自動的に勝ちになります。

■攻略のヒント

- 地形をよく見て自分のキャラが攻撃しやすいように配置しましょう。
- 攻撃のみに頼らず，地形を利用したダメージも計算に入れて戦ってみましょう。罠と直接攻撃を組み合わせることで，さまざまなバリエーションの攻撃が生まれます。

■そのほか

Expert MissionのQ&A，バージョンアップなどの仕様変更，インフォメーションなどはExpert Mission Officialホームページをご参照ください。

URL
http://www.antai.suginami.tokyo.jp/em/

ネットワークゲームの地平へ

CGIでの継続したユーザー管理

大和 哲 *Yamato Satoshi*

インターネットでのプログラミングといえば，PerlでCGI(Common Gateway Interface)を使ったものがもっとも手軽です。CGIを使えるサーバ自体が少ないのが難点ですが，リアルタイム性を要求されないゲームならば十分に記述できます。ここでは処理の基本となるユーザー管理の手順を見てみましょう。

■ゲームの方針

一般にWebサーバはPerlで書かれたCGIを実行する際に「Perlインタプリタを呼び出す」つまり，CGIが呼ばれるたびに別個にPerlタスクを起動させます。そのため，Perlで書かれたCGIの実行速度は遅くなり(実際には「実行速度」よりも「実行前段階」に時間がかかっているのが大きい)，またCGIの実行がサーバにとって「重い」負担になるのです。Webサーバソフトの代表であるApacheでは，mod_PerlというPerlインタプリタをApacheにモジュールとして組み込むことでPerlインタプリタがCGI実行ごとに呼び出される(起動される)ことを回避しています。これにより実行速度の向上とサーバマシンへのレスポンス改善が図られています。しかし，残念なことにWebサーバとしてのApacheのシェアは高いにもかかわらず，現在のところ，Apache+mod_Perlの組み合わせでWebサーバを一般に利用させているホスティングやプロバイダはまだ数えるばかりです。

そのようなわけで，複数の人間が，しかも少なくとも画面が書き換わるたびにアクセスされるような通信対戦ゲームをCGIで作るというのは，あまりおすすめできません。できれば，同じPerlでもサーバを1台立ててWebサーバから起動されるCGIではなく，専用のクライアントプログラムを作り，サーバ側とはdeamonプロセスとして独自にSocketを使って通信するのがよいでしょう。Perlでdeamonプロセスを作ること自体は大変簡単ですし，ほかの言語に比べるとメモリ管理も楽で，リークも起きません。ただし，作り方によっては，プロセスを無限増殖させてサーバを停止させてしまう，などということもありえますので厳重な事前テストが必要です。どうしてもWebブラウザ上で実行させたいなら，CGIではなくJavaアプレットなど，ほかの仕組みを使うほうがベターです(これはこれで制約はありますが)。

さて，「それでも」ということで，今回はCGIで対戦ゲームを作ります。繰り返していいますが，繰り返しリロードされるこの種のプログラムはサーバにとっては大きな負担となります。まず，サーバになるべく負担にならないような設計を考え，作ったプログラムは自分の環境で負荷が大きくなりすぎないことを確かめて，そのうえでサーバ上で試すようにしてください。

プログラムを書く前にまずどのようなゲームにするかを決めておきましょう。今回作成するのは，神経衰弱ゲームです。ただし，グラフィックも何もない簡単なゲームですが。

プレイヤーA，B2人がログインします。そして，ブラウザ上には，

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

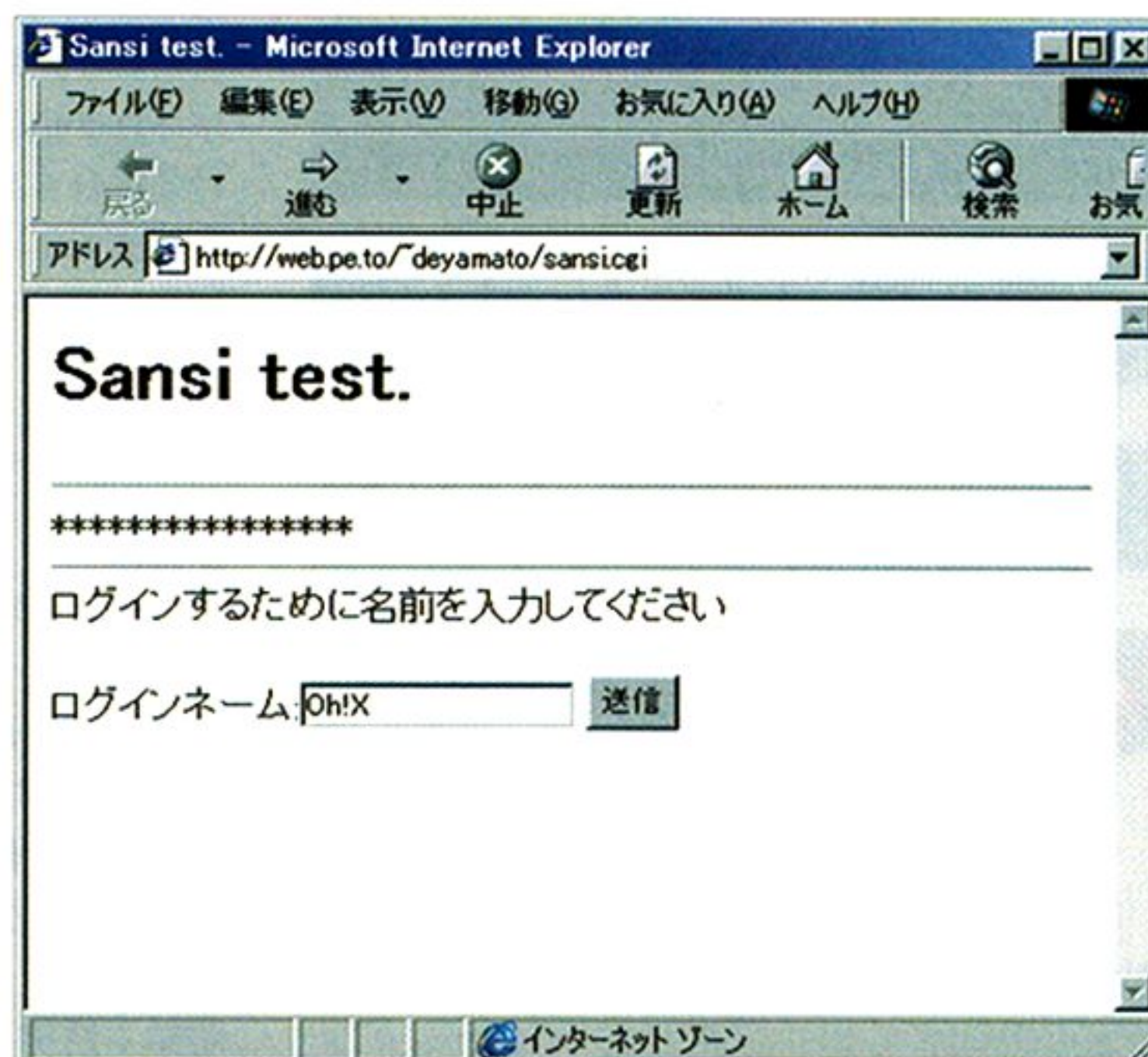
という16個のアスタリスクが並んでいます。それぞれのアスタリスクの裏には数字が隠れています。プレイヤーは＊を2つクリックします。その＊のところにあった2つの数字が同じであればもう一度そのプレイヤーは＊を2つクリックできます。違っていた場合はもうひとりのプレイヤーがめくります。そして，すべての＊が消えたらゲームは終了です。つまりは神経衰弱です。

この種のゲームプログラムを作るには，考えなければならないことがいくつかあります。そのなかでもいちばん大きなものは「プレイヤーをどのように管理するのか」でしょう。

図1　サンプルゲームをブラウザから起動

■ユーザーを管理するには

HTTPというプロトコルは，接続時間を短く保つために，もともと(HTTP1.0)は一度サーバに接続すると，ひとつのオブジェクト(ハイパーテキストやハイパーテキスト中のインラインイメージなどといったWebページを構成する部品)をやり取りするために，1回ごとにWebサーバへの接続/切断を繰り返すことが基本になっています(ただし，HTTP/1.1以降では改善されていますが)。

したがって，ユーザーが常に接続しているわけではないので，トランザクション処理のようなことを実現するのは非常にやっかいです。しかし，たとえばオンラインショッピングやゲームなどでそれぞれのユーザーがなにをしたかをトレースす

るためには，なんらかの方法で「いまアクセスしたのは誰なのか」ということを知り，それを内部のデータベースに結び付けなくてはなりません。アクセスしている人のうち，プレイヤーが誰であるのかをサーバが知るいちばん簡単な方法は，なんらかの方法でアクセスしたユーザー側のブラウザ自身がサーバに自分が誰であるかと名乗るという方法です。

サーバがユーザーを認識するためのいちばん簡単な方法は，ブラウザが初めてそのサーバにアクセスしたときにユニークなIDを与え，ブラウザがサーバにアクセスするときは必ず自分のIDを名乗るようにすることです。サーバ側はブラウザの名乗るIDをもとに処理を進めていきます。サーバがブラウザから送られるデータを読むには次のような方法が便利でしょう。

・GETメソッドで送られたQueryStringを読む

URLに添付してQueryStringとして送る方法は皆さんも見たことがあるでしょう。たとえば，チャットを行うCGIで，

```
http://foo.bar.com/cgi-bin/minichat.cgi
```

のようにブラウザのアドレス欄に入っているのに，名前やメッセージを入力すると，

```
http://foo.bar.com/cgi-bin/minichat.cgi?name=%70%8a%8d%……
```

というように本来のURLの後ろに，

```
?変数名1=…&変数名2=……
```

のようになにかメッセージを送っているようなものがありますね。これはGETメソッド(つまりブラウザがサーバにコンテンツの転送をリクエストする)パラメータを同時にQUERY_STRINGとして送っているのです。これを見ることでプレイヤーからの情報を得ることができます。

CGIプログラムからは%ENV{"QUERY_STRING"}という連想配列にその文字列が入ります。ブラウザが表示したHTMLドキュメントの中に<FORM>タグがMETHOD="GET"で指定されていた場合などは，

```
<INPUT NAME="変数名">
```

と書かれていると，この変数名がそのままURLの変数名として使われます。

ユーザーにハイパーリンクをクリックさせてその行動を追跡するには，ハイパーリンクが書かれたタグに，

```
<A HREF="./sansi.cgi?uid=NrpvewoYiHsAAADca6g">
```

というようなユーザーIDを書き加えておいて，アクセスされたときにその内容を読みにいけばユーザートラッキングが可能です。

なお，変数の内容はどんなバイナリなども送ることができて，REF1738で示されるエンコード方法で送られますので日本語の文字列などを送ることもできます(当然のようにコード変換などはされませんからその展開は別途考慮する必要があります。簡単なのはjcode.plなどですべて内部ではEUCコードで処理し，サーバからブラウザに送るHTMLのデータもcontent-type:x-euc-jpで送ることでしょうか)。

欠点として，この方法では環境変数の最大サイズよりも大きい文字列を送ろうとしても，途中で切られてしまうことがあります。

・POSTデータとして送る

GETメソッドと同じようにFORMを使ったデータのやりとりに使われているのがPUTメソッドです。POSTメソッドではGETメソッドと違ってCGIは標準入力ストリームを使ってデータの受け取りを行います。そのため，GETコマンドのように途中でデータが切れることがないのが特徴です(標準入力ストリームのデータを取って切り分けるのが面倒なときはcgi-lite.plやCGI:Liteモジュールを使うと楽です)。

また，送ったデータはブラウザのアドレス欄には表示されません。

HTMLの<FORM>タグがMETHOD=GETならGETメソッドで，METHOD=POSTなら，このPUTメソッドで処理されます(FROMではなく，単にURLにエンコードされている場合はGETになります。Oh!X復刊号で説明したようにWebブラウザがコンテンツをサーバから確保して表示するにはGETコマンドが使われているからです)。

ところで，皆さんは<FORM>タグとともに使う<input>タグのtypeにhiddenというパラメータがあるのをご存じでしょうか？　これは画面になにも表示させずに，value=で書かれた値がサーバに渡されます。つまり，このFORMの書かれたページ自体をCGIやSSIで書いて，

```
<INPUT TYPE=HIDDEN NAME="uid"
  VALUE="NrpvewoYiHsAAADca6g">
```

のようにvalue=にユーザーのIDを書いてやれば，サーバ側はこれを見てユーザーのトラッキングができます。

ですので，FROMでなにかを選ばせたり，記入させたりした場合にはこの方法も有効です。

■進行方法を考える

今回のゲームでのユーザートラッキングはURLにユニークなIDを組み込んで，サーバ側はQUERY文字列中のユーザーID(uid=)で識別することとします(注：ですのでゲームプレイの最中にはブックマークをしてはいけません。プレイヤーのユーザーIDまで一緒に登録されてしまうからです)。ただし，FROMでなにか記入させたデータを読み取るときは，POST形式でデータを渡させて読んでみます。

なお，これをするためには，ユーザーにユニークなIDをつける必要があるわけですが，このサンプルプログラムでは，このような用途にうってつけの$UNIQUE_IDを使用しています。これは，Apache1.3.0以降に固有の機能で，ユーザーがあるWebページをアクセスすると，アクセスごとに常にこの環境変数にユニークなIDを提供するものです。プレイヤーがこのゲームにログインしたときの$UNIQUE_IDをゲーム中のユーザーIDとして流用するわけです。ゲームが終わればこのIDは破棄されます(なお，Apache以外のサーバでも使えるように，この環境変数がなかった場合は適当に重なりそうにない文字列をランダムに作ってそれをIDにしています。文字列が長いのでぶつかることはないだろう……と思うのですが)。

では，ユーザートラッキングの問題が解決したところで，今度はゲームがどのように進んでいく

COLUMN ところでNetscapeCookieは？

ユーザートラッキングの方法として有名なのはNetscapeCookieを使う方法でしょう。GETコマンドやPOSTメソッドのように明示的に変数名と変数をユーザー側で設定できるわけではありませんが，ユーザートラッキングという意味ではNetscapeCookieもWebブラウザからサーバへデータを送ることができます……というか，ユーザーの知らないところで送られます。

Netscape CookieはまずサーバがWebブラウザにMETAタグやヘッダの一部として送られます。たとえば，CGIでーからヘッダの送信を行おうとすると，

```
prnt "content-type: text/html";
```

を送信すると思いますが，そのときに，

```
print "Set-Cookie: CONTENT=¥¥"変数＝値1;変数
                    ＝値2;……"
```

のような形で送信すると，NetscapeCookieに対応したブラウザはこのヘッダを見て，作業領域にこのクッキーのメッセージを仕舞い込みます。そして，再びブラウザがこのサーバをアクセスするときに，Cookieデータをサーバに渡すのです。CGIは$ENV{"HTTP_COOKIE"}という変数でこの値を参照することができます。

で，考えるまでもありませんが，アクセスしたときの状態をCookieに埋め込んでおけば二度め以降にその人がきたときには，前回の記録が環境変数に書かれているわけですから，前にいつきたのかなどを調べることが簡単にできます。また，Cookieは有効期限が設定されていて設定期間内しかPC内には保存されません。すでに過ぎてしまった有効期限を再送信すると削除することもできます。ただ，ゲームの場合，アクセス間隔が長い，たとえば一度アクセスして，次にくるのは何日後……というときでも使うことができるのは便利ですが，問題はこのクッキーを無効にするには日付を設定するしかない，ということです。

もちろんゲームのようなものがそれほど長期にわたることはないですが，一般的にゲームというものはいつ終わるかわかりません。たとえば，1ゲーム1分でゲームが終わるかもしれませんが，ひねくれたユーザーならタイムアウトぎりぎりまで次のアクションを待ってゲームをめいっぱい引き延ばしてしまうかもしれません。つまり，時間でクッキーを削除するのはあまり有効な方法とはいえないのです。

そのうえ，厄介なことにCockieの有効期限はサーバの時間ではなく，ブラウザのローカル時間での管理になっています。世界各地でクライアントがアクセスされる可能性を考えるとJST(日本標準時)だけで閉じておくわけにもいきませんし，そもそもブラウザが動いているマシンの時間が正しいと仮定して作る，というのも大きな冒険になってしまうでしょう。

ということで今回のゲームプログラムではNetscapeCookieは使っていません。

か，そのプロセスを考えていきましょう。CGIではゲーム中ずっとプログラムが実行されているわけではなく，ブラウザのアクセスごとに1回呼び出されるものなので，ここでstepという変数を用意して，この変数で現在のゲームの状況を示すことにします。そして，ゲームCGI中で，この変数を見て，その状況に応じて作業を実行し，作業を終了することにします。

次のステップでは，step変数の内容が変更されてまたCGIが呼ばれるわけです。よって，ここから下に書いたステップ1，ステップ2……という流れは一度に実行されるわけではありません。1回の呼び出しでステップひとつ分と考えてください。

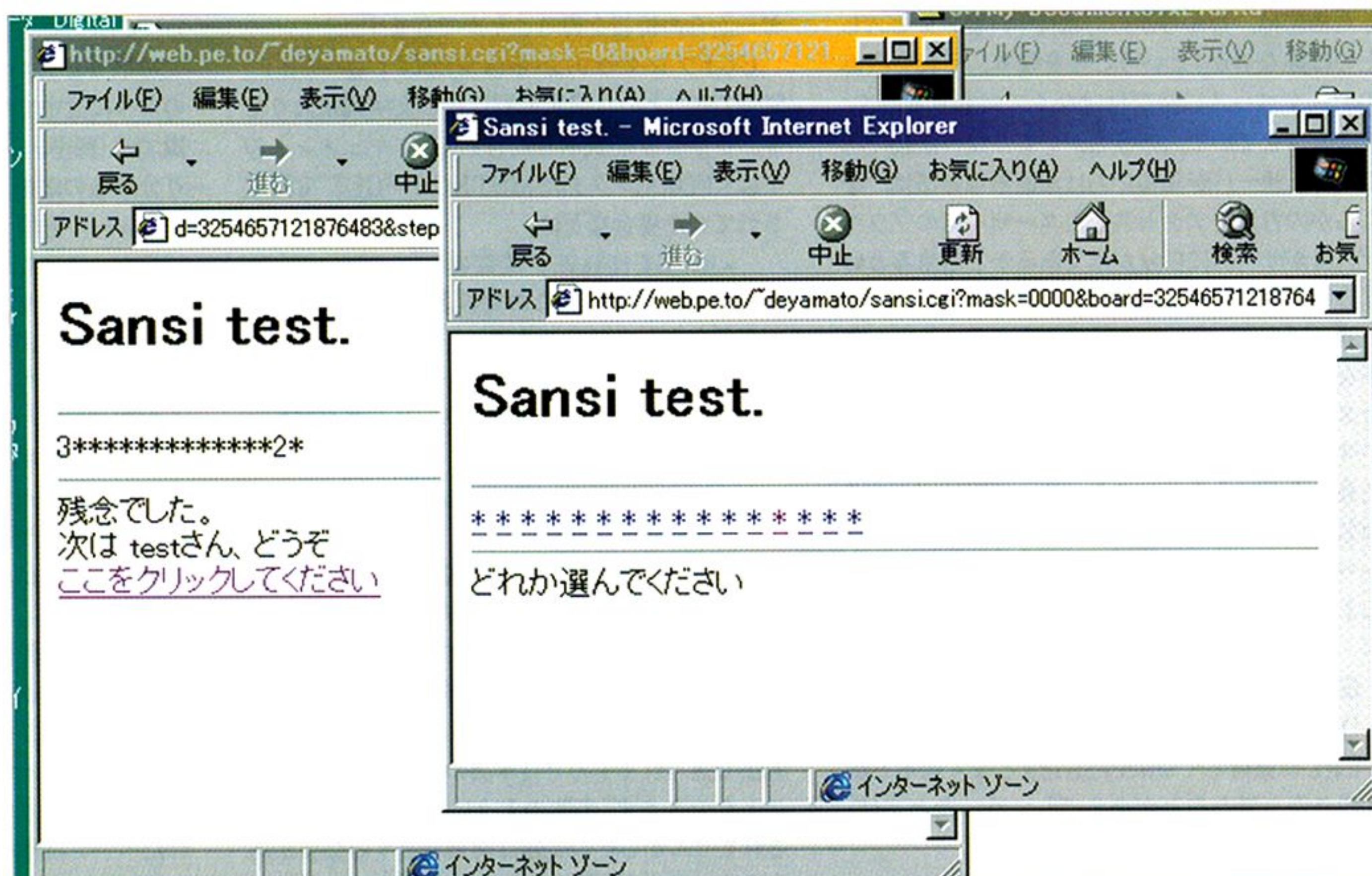



図2　ゲーム進行の様子

●ステップ1
ログインするために名前を入力させる
●ステップ2
ステップ3へ進ませるためのステップ
(ここで1ステップおかないとRefreshのたびにダイアログが開いてしまう)
●ステップ3
(これ以降自動リロードとなる)
　コマンド実行時刻を記録する
　プレイヤー1，プレイヤー2が揃ったらステップ4へ進む
　揃わなかったらこのステップで待つ
●ステップ4
もし，プレイヤー1，2どちらかがいないなら，
　「終了」
もし，自分のターンなら，
　どれかひとつ選ばせる
そうでなくて，
　もしマスクが$FFFFなら，
　「終了」
●ステップ5
2つめを選ばせる
●ステップ6
もし，プレイヤー1，2どちらかがいないなら
　「終了」
unmaskしたものの判定
もし，ひとつめと2つめが同じなら，
　print "おめでとう同じです
¥n";
　マスクの再設定
　選んだ2つのマスクを解除する
そうでないなら，
　print "残念でした。
¥n";
　もう一方のプレイヤーがこの状況を見るのを待つ
　プレイヤー交代
もし，もう一方のプレイヤーがこの状況を見たら，
　もし，マスクが$FFFFなら，
　　終了
　そうでないなら，
　　#--次のゲームへ
　　現在のプレイヤーのステップは4に

だいたい，このような感じになるかと思います。プレイヤー1は最初ステップ1，2，3，4，5，6と進んでいきますが，プレイヤー1を待たなければならないプレイヤー2はステップ1，2，3，4と進むと，プレイヤー1の進行をステップ4のまま待ちます(つまりステップ4の状態でこのCGIが何度も呼ばれることになるわけですね)。この繰り返し呼び出しの部分はhtmlのREFRESHタグによって一定間隔(15秒ごと)にブラウザが表示しているページのリフレッシュをかけて実現しています。

どのアスタリスクがクリックされたかは，

<A HREF="./sansi.cgi?id=0> * </A>

のようにURLのその*の位置を中に組み込みます。プレイヤーがこの*をクリックすると，ゲームCGIが起動し，?以降(Query文字列)を読み込み，id=0であればいちばん左のマスクをはずす，という処理をしています。

なお，このプログラムでは，マスクされている部分を0，はがされている部分を1として，その左右を逆に見て16進数で表しています。

　0000ならすべての数字にマスク，
　FFFFならすべての数字のマスクがはがれ，
　8000ではいちばん右の数字のマスクがはがれています。

また，マスクの下の数字なのですが，実はデバッグが簡単になるようにURLにはその内部表現が表示されています。URL上，board=と書かれているのがそれで，これも左右逆になっています。つまり，

　board=1122334455667788

ならweb上の，

　****************

に隠れているのは，

　8877665544332211

です。これは，Perlの「chop」という組み込み関数を使うと$_文字列の最後の文字を切り出してくれるという機能を使って文字列を16回ループを回し，それをprint関数で素直に表示するだけ，というプログラム上の構造によるものです。文字列の頭から簡単に切り出す関数があれば逆順にする必要もなかったのですが……。

■サーバ側はDBMにデータを記録

さて，ユーザートラッキングができたとして，サーバ側はユーザーのデータ，たとえばゲームに参加しているプレイヤーはuid誰と誰かや，board変数の中身などを持っておく必要があります。ユーザーIDがプレイヤー1のものならば，そのブラウザにはプレイヤー1用の，プレイヤー2にはプレイヤー2の画面を，それ以外の人にはそれにあったデータを表示させなければなりません(ちなみに，このゲームではプレイヤー1，2以外はログインすることができませんが，現在のゲーム状況を見ることはできるようになっています)。

このゲームCGIでは，これらの情報が連想配列に入っているので，扱いが簡単なDBMファイルというかたちでサーバ上に確保することにし，これを読み書きすることでサーバ側のユーザー管理

を行うことにしました。

このDBMファイルの説明を少ししておきましょう。DBMファイルとはデータベース管理ファイルとも呼ばれ，もともとUNIX系のOSでユーザーパスワード管理をするときに便利なように作られたファイル形式なので，Perl5でもWin32用のPerlからは使うことができません。

扱い方はとても簡単です。

Perlには連想配列という，簡単にいうと「文字列を添字に使える配列」というものが存在しています。たとえば，

```
my %players
```

これでplayersという名前の配列を宣言して，

```
$players{"player1"}="AAA";
$players{"player2"}="BBB";
```

のように変数を使うことができ，

```
print $players{"player2"};
```

を実行するとBBBが表示されるわけですね。すでにCGIを作っている人ならばお馴染みのPerlの機能だとは思いますが。この連想配列1個を簡単にひとつのファイルにすることができます。

この機能はDB_Fileモジュールに入っていますので，使うには，

```
use Fcntl;
use DB_File;
```

を宣言しなければなりません。

この連想配列をDBMファイルを結びつけるにはtie，untieという関数を使います(このtie，untieでDBMファイルと連想配列を結びつける機能はPerl5以降で追加されたものです)。

```
tie%players,DB_File,"players.dbm",
O_RDWR,0666
```

これで%players配列をplayers.dbmというファイルに結びつけることができます。この時点でplayers.dbmファイルに変数の内容が入っていれば%players配列の中にその内容がコピーされるわけです。

この関数のパラメータ中でのDB_Fileはパッケージ名で，DBMファイルを使うときは常にDB_Fileとなります。O_RDWRはリードライトモードを示し，読み込みのみのモードはO_RDONLYになります。その後ろの4桁の8進数はお馴染み，UNIXのパーミション指定です。もし，ファイルを新規に作成するときにはこのパーミションでファイルが作られるわけです。

そして，

```
$players{"player1"}="AAA";
$players{"player2"}="BBB";
```

のように配列中の変数に値を入れて，

```
untie % players
```

でDBMファイルに連想配列の内容がディスクに書き込まれます。

もし，DBMファイルが使えないようなプロバイダを使っている場合は普通のファイルで代用してもかまわないでしょう。大きなファイルを扱うとファイルの読み書きに時間がかかること，ファイルロックを注意しないとファイルが消えることなどがある点には気をつけなければなりませんが。

逆にRDBMSを使ってみるというのも面白いかもしれません。最近はMySQLやPostgreSQLが使えるホスティング/レンタルサーバというようなものもあります(http://www.he.netなど)のでそういう会社のレンタル領域を借りてみるといいかもしれません。

リスト1　viewdbm.pl

```
#!/usr/local/bin/perl

use Fcntl;
use Getopt::Long;
use DB_File;

GetOptions( "d|dbmname:s" );
my $dbmname = $opt_d || die  "Usage :$0 -d dbmfile\n" ;
my %contents;

tie %contents,DB_File,$dbmname,O_RDONLY,0600,$DB_HASH || die  "Couldn' t open DBM" ;

$FORMAT_LINES_LEFT=0;

my $i=0;
while(($number,$record) = each %contents){

          print  "[$i] $number = $record \n" ;
          $i++;

}

untie %contents;
```

リスト2　clrdbm.pl

```
#!/usr/local/bin/perl

use Fcntl;
use Getopt::Long;
use DB_File;

GetOptions( "d|dbmname:s" );
my $dbmname = $opt_d || die  "Usage :$0 -d dbmfile\n" ;
my %contents;

tie %contents,DB_File,$dbmname,O_RDWR,0600,$DB_HASH || die  "Couldn' t open DBM" ;

$FORMAT_LINES_LEFT=0;

undef %contents;

if (1==untie %contents){
          print  ".dbmの削除に成功しました\n" ;
}else{
          print  ".dbmの削除に失敗しました\n" ;
}
```

■プログラムについて

ということで，できあがったのが付録CD-ROMに収録されたサンプルプログラムsansi1.cgiです。これまで説明してきたように神経衰弱ゲームです。プレイするにはこのsansi.cgiを実行して，2名でログインしてください(同じマシンからでも別のマシンからでもかまいません)。2人ともログインしたらゲームがスタートします。

それから，本文中では解説していませんが，プレイヤー1，2のアクセスがいつ行われたかが記録されており，最後のアクセスから1時間超えるとタイムアウトとなり，それまでプレイヤー1，2のuidを持つブラウザがアクセスしてもゲーム終了が表示されるようになっています。

また，リスト2，3は任意のdbmファイルの連想配列の内容を見る(viewdbm.pl)，内容をクリアする(clrdbm.pl)スクリプトです。telnetでWebサーバに入って使用してください。

SPECIAL

特集 ネットワークゲームの地平へ

Macintoshをサーバにしてゲームを作ろう

古籏一浩 *Furuhata Kazuhiro*

昨今では個人ベースでWebサーバを持つことも夢物語ではなくなってきました。個人ベースの場合もUNIX系のWebサーバが多いのですが，Macをサーバにしてしまうことも不可能ではありません。ここではMacをゲームサーバとして立ち上げて，CGIでゲームを作るまでの手順を見ていきます。

■MacintoshでWWWサーバ

WWW(World Wide Web)サーバというとUNIXやWindowsNTで動くものといった感じがありますが，MacintoshにもWWWサーバソフトがあります。特にMacOS8.xからはOSに標準で入っているためTCP/IPの設定をしてしまえば「コントロールパネル」内の「Web共有」を選択し「開始」ボタンを押すだけでWWWサーバが起動します。

Macintosh用のWWWサーバは標準のもの以外にもたくさんあります。シェアウェア(Netpresentz/MacHTTP[*1]など)もあれば市販のもの(WebSTARなど)もあります。今回はMacHTTPを使います。特にMacOS8.xでなくても動作しますし，ダブルクリックするだけで手軽ですしCGIにも対応しています。

MacintoshでWWWサーバを構築した場合のCGIプログラムの呼び出しはAppleEventで行われます。このAppleEventを受け取ることができるプログラム環境があればCGIプログラムを構築できます。CGIを処理できる開発環境はいくつかありますが，現時点では以下のようなものがあります。

・Code Warrior(Pro)
　C/C++開発環境です。AppleEvent周りは自前で作成する必要があります。その代わりあらゆる処理が行えます。出費は6万〜7万円程度必要です(学割はあります)。

・Future BASIC(II-J)
　BASIC開発環境です。C/C++同様にAppleEvent周りは自前で作成する必要があります。Future BASICでCGIを作成するのは，あまりメリットがありません。出費は3万〜4万円程度必要です。

・REAL Basic
　AppleEventの扱いが非常に簡単です。日本語版も出ますので利用価値は十分あります。日本語版の値段は現在不明です。

・AppleScript
　MacOSに標準で入っています。日本語も使用でき出費はありません。難点は速度でしょうか。

・MacPerl(ver.5)
　フリーウェアです。MacJPerlもあります。インターネット上からダウンロードできます。出費は電気代/電話代。雑誌のCD-ROMに収録されている場合がありますので，それらを利用するのもよいでしょう。幸いUNIXのPerlとほぼ同じ(UNIX特有のものは駄目)ですので，既存のスクリプトが流用できます。Perlの参考書はたくさんありますが，いずれもUNIXベースでありMacPerlベースのは1冊もないようです。

今回はMacJPerl ver.5(今回は2バイト処理はしないのでMacPerl ver.5でも同じです)とMacHTTPという組み合わせで簡単な「数あてゲーム」を作ってみます。基本部分がわかれば，あとは応用ですからシミュレーションゲームでも，RPGでも，アドベンチャーゲームでも作成できるはずです。さすがにリアルタイムゲームは無理ですが。

■サーバを起動する

まずMacHTTPを入手します。インターネット関係の雑誌の付属CD-ROMから入手するか，http://www.biap.com/あたりから入手します。市販されているWebSTARでも大丈夫だと思います。

次にMacPerlを入手します。同様に雑誌から入手するかftp://ftp.lab.

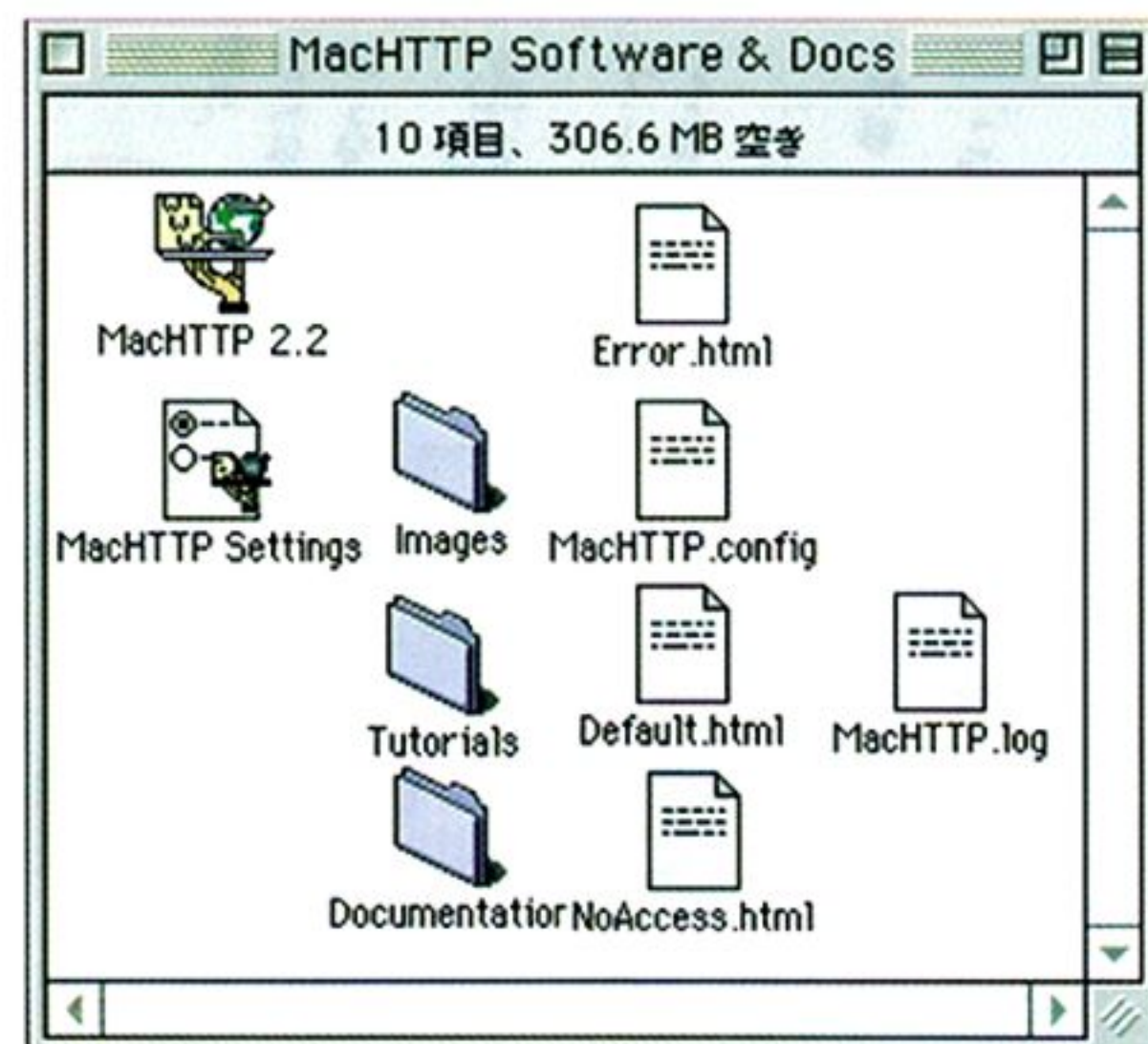



図1　MacHTTPのフォルダ内容

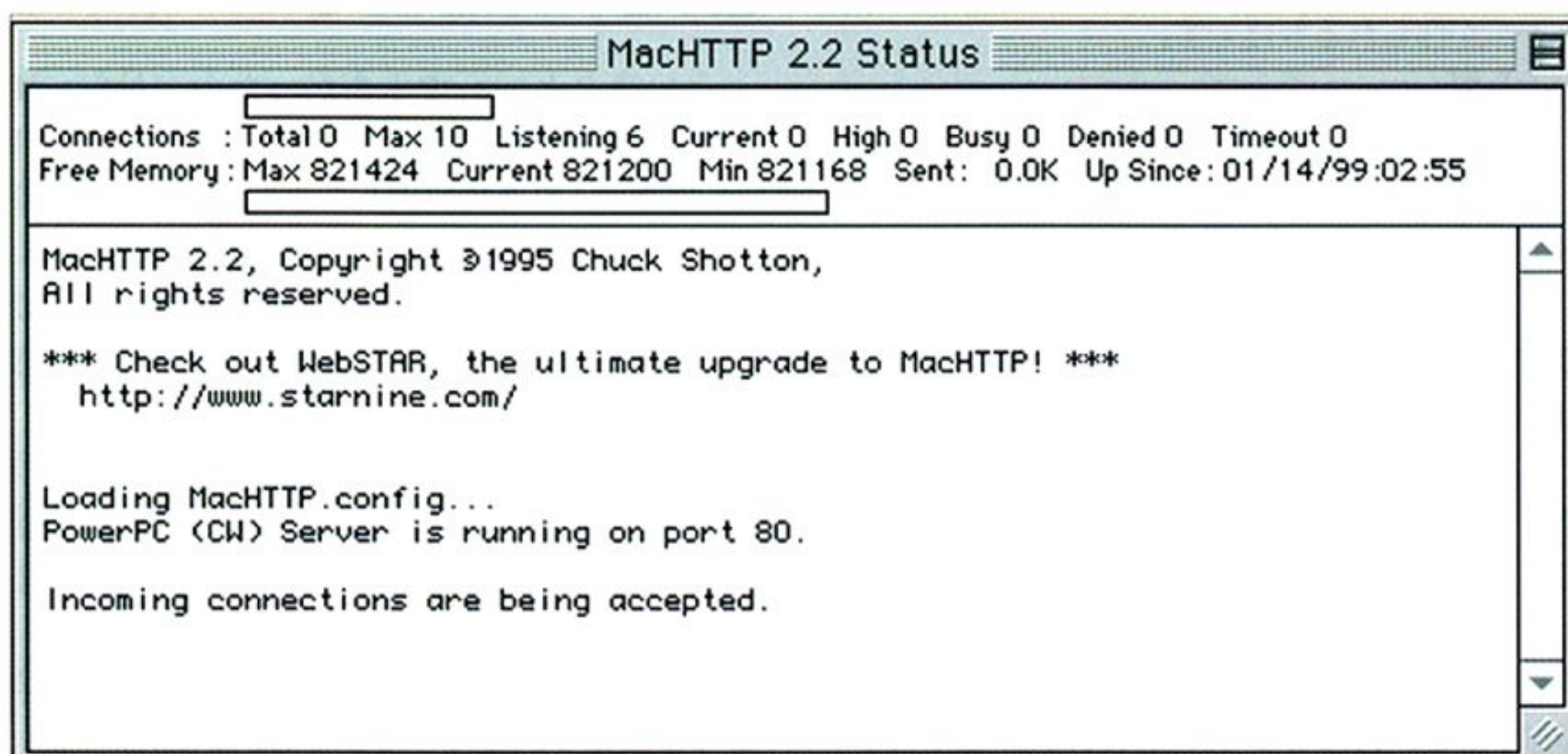



図2　起動するとこのような画面になる

＊1　MacHTTPはWebSTARの前身です。

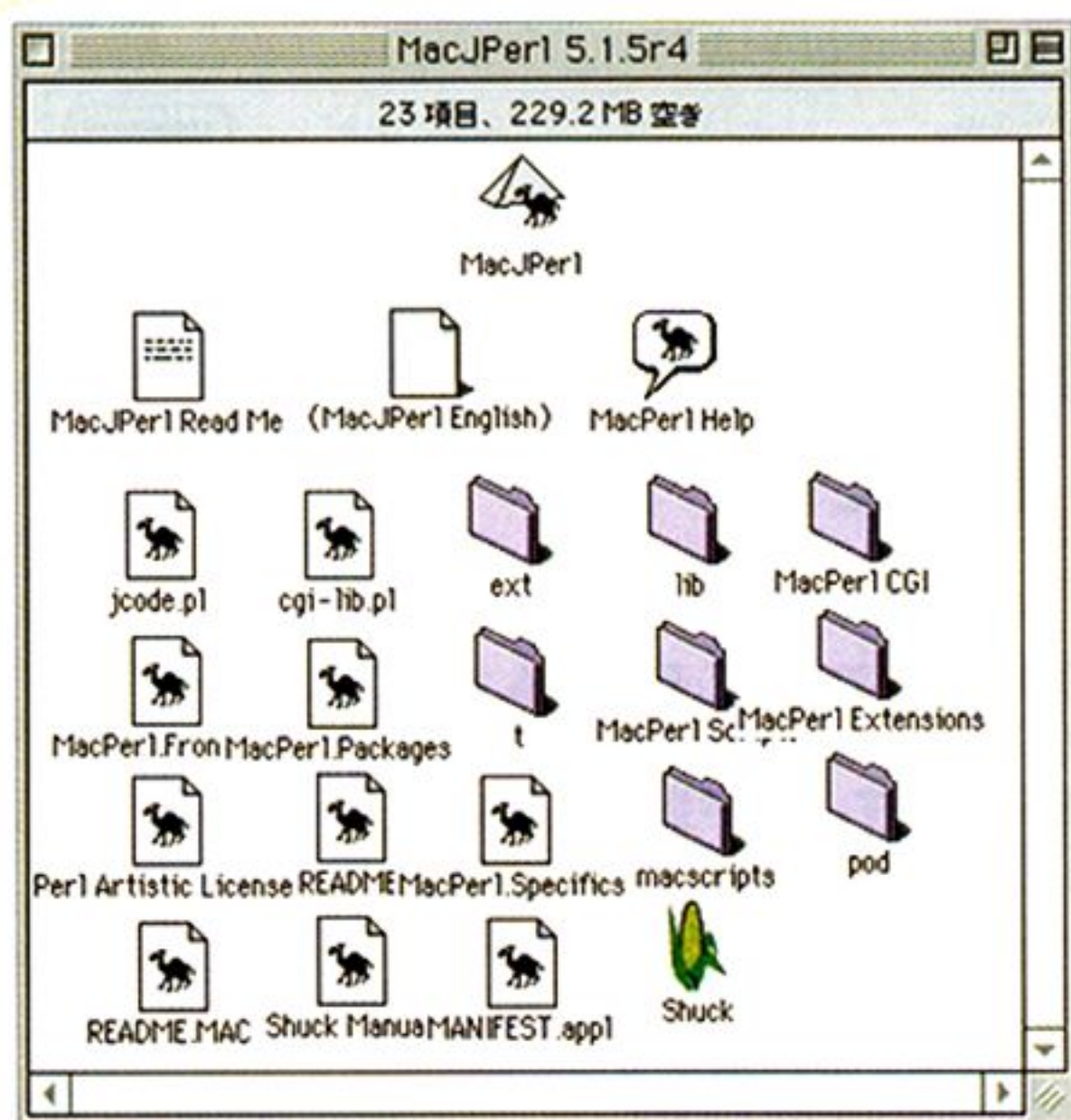



図3　MacJPerlのパッケージ内容

図4　MacJPerlの様子

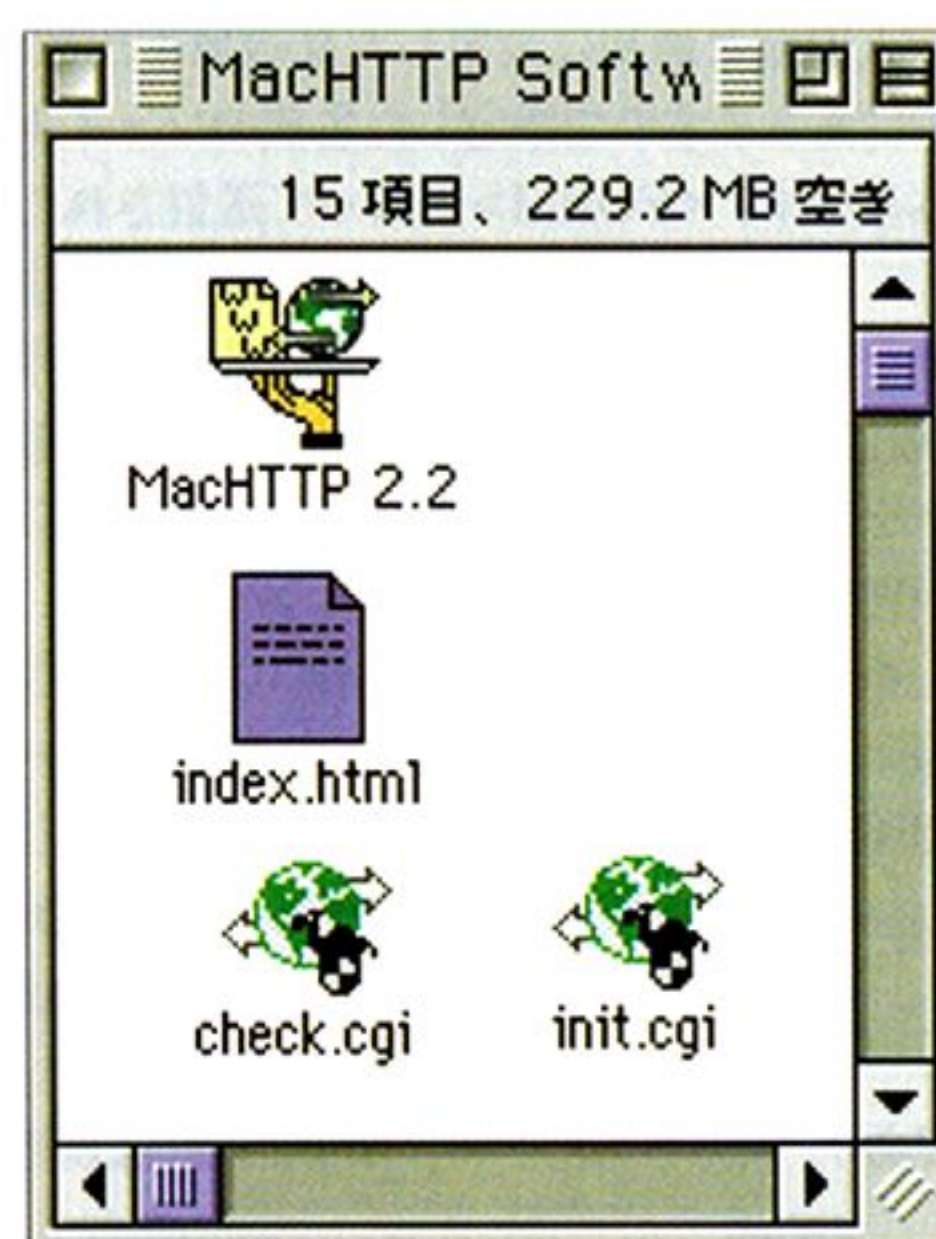



図6　Macでも拡張子は忘れずに

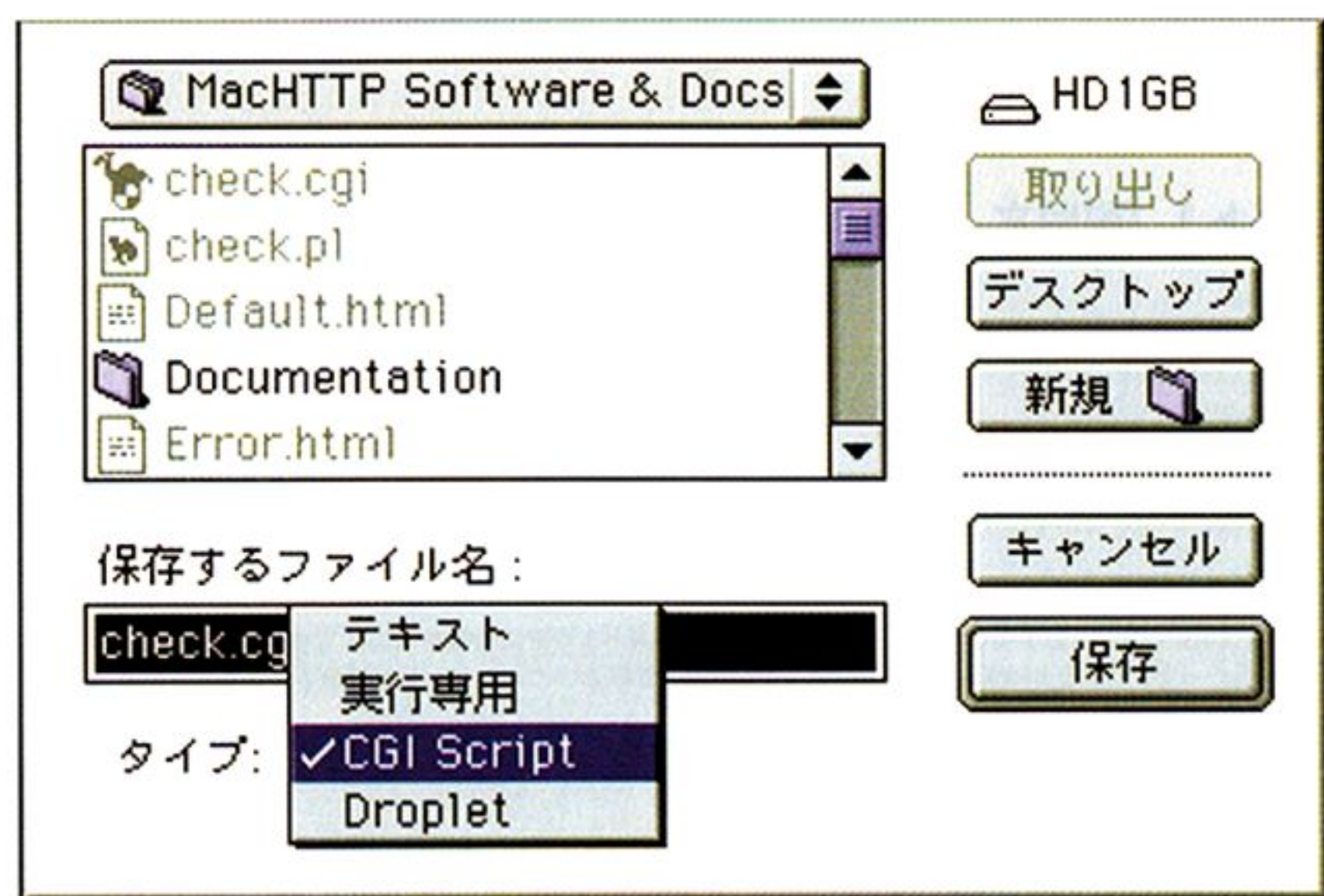



図5　CGIスクリプトとして保存すること

kdd.co.jp/lang/perl/CPAN/ports/mac/からダウンロードしましょう。

圧縮されている場合はStuffIt/StuffIt Expanderなどで解凍してください。次にTCP/IPの設定を確認します。専用線でなくてもテレホーダイ時間に接続しっぱなしであれば，割り当てられたIPを調べて告知するだけで，暫定的な(深夜だけの)サーバができあがります。

TCP/IPの設定を確認したらMacHTTPを起動します。図1の中のMacHTTPアイコンをダブルクリックすれば図2のような画面になりWWWサーバが起動します。

MacHTTPの設定(ポートやMIME Type)を行う場合はMacHTTP.configファイルをSimpleTextやJEDITなどのテキストエディタで開いて修正します。ほかにサーバソフトを動かしていないのであれば特に問題ないでしょう。FileMakerサーバなどを使用している場合はポート番号を変更するか，FileMakerサーバ側の設定(ポート)を変更してください。

これで準備はOKです。次はMacPerlです。

■MacPerlでCGIプログラムを作る

ここではMacJPerl ver.5を使用しています。MacJPerlのフォルダ内容は図3のようになっており，MacJPerlアイコンをダブルクリックすると起動します。プログラムを入力するにはファイルメニューから新規を選択します。ウィンドウが1枚表示されますから，そこにプログラムを入力していきます(図4)。

入力し終わったら，Command＋Shift＋Rキーを押して実行です。

実行前に文法チェックも行えます。文法チェックを行う場合はCommand＋Shift＋Kを押します。エラーがある場合はエラーメッセージが表示され，エラー行が自動的にハイライト表示されます。

エラーがなければファイルを保存します。とりあえず，テキスト形式で保存します。この状態では作成したプログラムはCGIとしては認識されず，使用することができません。ここがUNIX/Windowsなどと違うところです。MacPerlで作成したスクリプトをCGIとして認識させる場合は，保存する際に保存タイプを「CGI Script」にする必要があります(図5)。

CGI Scriptとして保存しても入力したプログラムは保持されておりMacPerlのファイルメニューから「開く...」を選択することで再度編集することが可能です。

次にMacHTTPで扱うCGIの場合はCGIファイル名の末尾(拡張子)が.cgiとなっていなければなりません。たとえばファイル名が「sample」だけでは駄目で「sample.cgi」のようにする必要があります。作成したCGIスクリプトはMacHTTPフォルダ内に入れておきます。今回のサンプルはルート(大元のフォルダ)に入れました(図6)。

その他のフォルダに入れる場合は正しいパス(フォルダの位置)を<FORM>タグのACTIONで指定する必要があります。

これで完了です。

■プログラムの概要

プログラムは初期化部分とチェック部分の2つに分かれています。初期化部分で乱数を発生させファイルに保存します。srand(time)が乱数を発生させるためのシード(乱数の元)を指定する部分です。rand命令で乱数を発生させファイルに書き込みを行います。MacJPerlで排他ロック命令flockを使用すると未実装と表示されてしまい使えないので排他処理はしていません。あとはHTML文書を吐き出して終了です。

チェック部分は入力された値を比較し対応したメッセージを出力します。メッセージの出力はヒア文字列を使って出力しています。print命令で書いても構いません。ヒアドキュメント中に$xxxxとあれば該当する変数の値に変換して出力してくれます。たとえば，入力した値を表示させる場合は以下のようになります(変数$myValに値が入っているものとします)。

```
print "入力された値は $myVal です";
```

もし$myValの値が6であれば，

```
入力された値は6です
```

と表示されることになります。

入力値と最初に設定した乱数値と比較する部分ですが，乱数値はファイルにありますので指定した名前のファイル（kazu.txt）をオープンし1行読み込みます。<FORM>によって送信された値を受け取り処理するには，

```
read STDIN,Scheck,$ENV{'CONTENT_LENGTH'};
```

となります。STDINは標準入力からのデータ，つまり<FORM>で入力（送信）されたデータです。$ENVは環境変数といっていろいろな環境データを返します。MacPerlでは無視されてしまう環境データも結構ありますが実用としては問題ないでしょう。

<FORM>タグのMETHOD="POST"の場合は$ENV{'CONTENT_LENGTH'}とすることで入力されたデータの長さを知ることができます。<FORM>で入力されたデータですが，

```
<FORM METHOD='POST' ACTION='check.cgi'>
<INPUT TYPE='TEXT' NAME='val'>
</FORM>
```

であり，入力されたデータが6であれば以下のようなデータが送られてきます。

```
val=6
(名前=値)
```

つまり<INPUT>タグのNAMEオプションで指定された名前と値を=で区切ったものが送られてきます。これはvalと=と6に分ける必要があります。

```
(Sdummy,Scheck) = split /=/,Scheck
```

として$checkに入っているデータを分解します。

PerlもBASIC同様サブルーチンが使用でき，

```
sub サブルーチン名
{
      処理内容
}
```

というように定義することができます。呼び出す場合はサブルーチン名に&をつけて呼び出します。たとえばohmzという名前のサブルーチンであれば，

```
&ohmz
```

として呼び出すことができます。

■最後に

Perlでのプログラムは超初心者なので，全然Perlらしくありません。あまり説明になっておらず申しわけありません。プログラムのほうもエラー処理が甘く入力値が数値かどうかもチェックしていません。不具合部分はPerl本を参考にして改良してみてください。

とりあえず数あてゲームができて，原理がわかればシミュレーションや育てゲームが作れるでしょう。Macintoshだとパーミッションとかパスとか悩まなくてもよいので手軽にPerlプログラムを作成することができるのではないかと思います。ちょっとしたネットワーク上でゲームを稼働させるにはちょうど手ごろでよいかもしれません。

今回作成したプログラムは以下のアドレスで実行できます。お試しください。

http://www2.shiojiri.ne.jp/~openspc/game/kazu/

最後にMacPerlの注意点を載せておきます。

＊注意点

MacPerlはUNIXのPerlなどと異なり1行目のPerlスクリプトのパス（フォルダの場所）を指定する部分は無視されます。また，セキュリティですがパーミッションなどは特にありません。好きにファイル作成/削除/書き込みができます。CGIとして機能させる場合は保存時に保存タイプを「CGI Script」として保存しないと機能しません。flockの排他処理は駄目みたいです。chmodなどUNIX独自のものは当然使用できません。

図7　今回作成したゲームの様子

リスト1（説明文／HTML）

```
<HTML>
<HEAD>
<TITLE>Game</TITLE>
</HEAD>
<BODY bgColor="#FFffFF">
<CENTER>
<H2>数あてゲーム</H2>
<BR>
<BLOCKQUOTE>
  1～10までの数を当てるゲームです。フォームのテキストエリアに数値を入力してボタンを押します。値
  が当たっていれば正解の表示，そうでなければ大きいか小さいのメッセージが表示されます。
</BLOCKQUOTE>
<FORM METHOD="POST" ACTION="init.cgi">
<INPUT TYPE="submit" VALUE="ゲームを始める">
</FORM>
</BODY>
</HTML>
```

リスト2（初期化部分）

```
#!
#*******************************************
#  乱数を生成しファイルに保存するスクリプト
#*******************************************
#====== 乱数生成しファイルに書き込み =======
$filename = ">kazu.txt"; #flockが使えるなら$filename = ">>kazu.txt";
$errorMessage = "File open error...";
$filelock = 2;
srand(time|$$);
open(KAZU_FILE,$filename) or die $errorMessage;
#flock KAZU_FILE,$filelock; #flockが使えるなら外す
#truncate(KAZU_FILE,0); #flockが使えるなら外す
print KAZU_FILE int(rand(10)) + 1;
close KAZU_FILE;

#============= HTML文書を出力 ==============
print <<"__HTML__";
Content-type: text/html\n\n
<HTML>
<HEAD>
<TITLE>数あてゲーム</TITLE>
</HEAD>
<BODY bgColor='#FFffFF'>
<CENTER>
<H2>数あてゲーム</H2>
<BR>
<FORM METHOD='POST' ACTION='check.cgi'>
1～10の値を入れて：<INPUT TYPE='TEXT' SIZE='4' MAXLENGTH='2' NAME='val'>
<INPUT TYPE='submit' VALUE='押すべし'>
</FORM>
</BODY>
</HTML>
__HTML__
__END__
```

図8　適当な数を入れて……

図9　大小比較で正解を探していく

リスト3（チェック部分／メイン）

```
#!
#******************************************
#      判定および結果出力スクリプト
#******************************************
#------------------------------------------
#  メッセージ表示（小さい）
#------------------------------------------
sub little
{
print <<"__HTML__";
Content-type: text/html\n\n
<HTML><HEAD><TITLE>数当てゲーム</TITLE></HEAD>
<BODY bgColor='#ffFFff'>
<center><H2>外れだよ</H2>
$checkよりも<B>小さい</B>みたいです。
<BR>
<BR>
<FORM METHOD='POST' ACTION='check.cgi'>
1～10の値を入れて：<INPUT TYPE='TEXT' SIZE='4' MAXLENGTH='2' NAME='val'>
<INPUT TYPE='submit' VALUE='押すべし'></FORM>
</BODY>
</HTML>
__HTML__
}

#------------------------------------------
#  メッセージ表示（大きい）
#------------------------------------------
sub large
{
print <<"__HTML__";
Content-type: text/html\n\n
<HTML><HEAD><TITLE>数当てゲーム</TITLE></HEAD>
<BODY bgColor='#ffFFff'>
<center><H2>外れだよ</H2>
$checkよりも<B>大きい</B>みたいです。
<BR>
<BR>
<FORM METHOD='POST' ACTION='check.cgi'>
1～10の値を入れて：<INPUT TYPE='TEXT' SIZE='4' MAXLENGTH='2' NAME='val'>
<INPUT TYPE='submit' VALUE='押すべし'></FORM>
</BODY>
</HTML>
__HTML__
}

#------------------------------------------
#  メッセージ表示（正解）
#------------------------------------------
sub hit
{
print <<"__HTML__";
Content-type: text/html\n\n
<HTML><HEAD><TITLE>数当てゲーム</TITLE></HEAD>
<BODY bgColor='#ffFFff'>
<center><H2>当たり！</H2>
正解した値は$kazuです。<BR>
再度チャレンジする時は<A HREF='index.html'>ここ</A>を押してね(^^)/
</BODY>
</HTML>
__HTML__
}

#------------------------------------------
#  メッセージ表示（エラー）
#------------------------------------------
sub error
{
print <<"__HTML__";
Content-type: text/html\n\n
<HTML><HEAD><TITLE>数当てゲーム</TITLE></HEAD>
<BODY bgColor='#ffFFff'>
<center><H2>入力された値が変です</H2>1から10までの数字を入れてね。<BR>
<BR>
<BR>
<FORM METHOD='POST' ACTION='check.cgi'>
1～10の値を入れて：<INPUT TYPE='TEXT' SIZE='4' MAXLENGTH='2' NAME='val'>
<INPUT TYPE='submit' VALUE='押すべし'></FORM>
</BODY>
</HTML>
__HTML__
}

#------------------------------------------
#  メインルーチン
#  入力された値と正解値を比較処理
#------------------------------------------
#====== ファイルとFORMからの読み込み =======
open(KAZU_FILE,"kazu.txt")
    or die "File read error or not found...";  # 正解値が書き込まれたファイルを開く
$kazu = <KAZU_FILE>;                            # ファイルから1行入力
close KAZU_FILE;                                #  ファイルを閉じる
read STDIN,$check,$ENV('CONTENT_LENGTH');       #  フォームで入力されたデータを変数に代
入
($dummy,$check) = split /=/,$check;             #  名前と値を分解し値を$checkに代入

#============== 判定処理 ==================
if (($check < 1) || ($check > 10))
{
    &error;
}else{
    if ($kazu == $check)
    {
        &hit;
    }else{
        if ($kazu < $check)  { &little;}else{ &large;}
    }
}

__END__
```

特集 ネットワークゲームの地平へ

ネットワーク対戦エミュレータBot

須藤芳政 *Sudo Yoshimasa*

ネットワーク対戦ゲームの王道といえばQuake2。対戦ゲームはいいのですが，サーバに誰もログインしていないとひとりでは対戦ゲームになりません。そんなときのために自動対戦プログラムが存在します。Botプログラムはネットワーク対戦でプレイヤーの代わりとなるものなのです。

今年の冬，私は新しい毛布と布団を購入し，さらに進化した生活を送ることができました。もう洗濯物を全身に被って震えながら眠るようなみじめな私ではありません。今後のプランとして綿が飛び出してボロボロ状態の東京◯野市のゴミステーションから拾ってきた「高級羊毛敷布団」(そう書いてあった)を新調することを考えております(拾う手段含む)。

「貴様！　贅沢がすぎるんじゃないのか？」

という声もあるかもしれませんが，贅沢させていただきますぞ私は！　たとえば……ん～と，宅配ピザ注文とかコンビニで神田川俊郎弁当買ったりとかしちゃうぞコンニャロ！

今回はQuake2のプログラミングについてお話ししたかったのですが，「あること」に年末年始からひたすら没頭していたせいで，皆様にお届けするネタが十分に用意できなかったのです。その「あること」とはUnrealやSin，Shogo，Half-Lifeで遊び呆けていたことではありません(このせいじゃないのか？)。ズバリ，「あること」とはBotのことです。

■Botってなに？

この前，実家から電話があったとき「アンタまだ寝てるの？」という母の問いに「Bot作ってんだよ」と答えたところ，「なに，ボットって？」といわれたので，「新宿で流行ってんだよ」と答えたら「それより部屋ちゃんと掃除しないと大家さん困るでしょ！」といわれました。世の親はBotに無関心です。

Botを辞書で検索してみると「ウマバエの幼虫」とか「ウマバエの幼虫が寄生して起こるボッツ症」なんて書いてありますけど，私はウマバエ研究に年末年始の時間を費やしたわけではありません。Botっていったいなんなんでしょうか？

まずはBotの語意や語源よりも先に私が今回のテーマとしているネットワーク対戦におけるサーバサイドのBotとはどのようなものであるのか説明してしまいましょう。

現在はネットワーク対戦に対応したゲームが多数存在しますが，ゲームサーバ側で接続している各プレイヤー間の情報更新タイミングを調節しても太くて速い線で接続しているプレイヤーが有利であることに変わりはありません。各家庭からゲームサーバへ殴り込みをかける場合，一般的には電話回線とプロバイダという二重の問題があるので，

「いやあ，今日からISDN引いたんで，サーバで大暴れさせてもらいますよ～」

なんて意気込んでも，実はプロバイダの線が激ノロだったので，

「おめ，全然変わんねえじゃんかよ！　うがー！」

と怒り狂うこともよくあります。

私も56Kモデムを一大決心で購入し，

「いやあ，今日からは大暴れさせてもらいますぞ～」

なんて思っていたら，プロバイダがK56flexとかいうプロトコルにしか対応していなかったので買ったモデムで繋がるのは最大33.6kbps。

「おめ，ぜんっぜん変わんねえじゃんかよ！　うっがー！」

と怒り狂いました。誰か！　このモデム買い取ってク・レ・ヨ！

といった具合に，回線による問題に頭を抱えている人は多いはずです。そんな人たちの救世主がBotなのです。

Botは本物のプレイヤーの代わりに貴方の相手をしてくれます。つまりネットワーク対戦のエミュレートをローカルマシン上で行えるので，ほかのプレイヤーとの回線の速度差など気にする必要がないのです。もうわかりましたか？　Botはロボットのことです。

Botの使い道はほかにもあります。LANを使って友達と対戦を行う際，人数が少ないのでイマイチ盛り上がりに欠ける場合にもBotを2，3体参加させるとよいでしょう。ネットワーク対戦初心者が初陣でボコボコにされ，復讐心に燃えて鍛練に励む場合にも役立ちます(このあとの悲惨な話を読むと「なんだ，結局役に立たないんじゃないか」と思うかもしれませんけど)。

ロボットと聞いて思い出しました。榊原郁恵の「ロボット」(だったような気がする)。いまじゃ怪傑熟女仲間の彼女が昔，スターウォーズのC3-POを倍速にしたような奇怪な踊りをしながら歌って

3rd-Zigock II Guide - Microsoft Internet Explorer

ファイル(F)　編集(E)　表示(V)　移動(G)　お気に入り(A)　ヘルプ(H)

中止　更新　ホーム　検索　お気に入り　履歴　チャンネル　全画面表示　メール

アドレス　G:¥x2cd¥1¥quake2¥3ZB2¥3zbII Users Guide.htm

3rd-Zigock 2 Guide For V0.80

Writen By 006.9 [RsF] to allow users to get the most out of this "Outstanding Bot"

Bot Author: ponpoko E-mail add. : ponpoko@axcx.com

3zbII is a multiplayer simulator for offline practice or for those who have such a bad pings/lag which make playing online near imposible to play ... At the presant time only Standard Quake2 and The Reckoning addon Deathmatches are supported ... Bots as they are known are simulated human oponents that will move around the map picking up and useing the same weapons as you, on you and on any other Bot in it's sights ... They can Rocketjump to! so you'll have competition for the megahealth on q2dm1

This 5 Step guide will show you how to set up the Deathmatch settings, maps, bots and start a game...

1. Installing 3zbII
2. Installing the Reckoning Support **(requires 91mb Free Space)**
3. Setting up the Server Game.cfg
4. 3zbII Commands
5. Runing 3zbII and adding bots

Advanced use of 3zbII

- Tweaking the bots
- Making new routes

In Development ...

- Where to find Updated 3zbII virsions when released

Links ...

- *The UnR Jump Center* ... Tips and know how on jumping and manuvers ... Execellent Site
- *Planetquake* ... Player Models Skin Maps and many other Quake and Quake2 related stuff
- *Quake II Bot Emporium* ...
- *id Software* ...
- *3M Precise Mousing Surface* ... People spend a lot on cash on gfx cards and other things to boost their perfomance and yet overlook an important part of their system ... contol and most often a nasty mouse mat that lets player performance down ... well heres the mat to have and it only cost about £10 - £12 so it's not much to pay !

マイコンピュータ

図1　新型Botのマニュアル。英文なのは当然か

いました。興味のある方は中古レコードショップで探してみるのもいいかもしれませんね。私は興味ありませんけど。

■Botとの出会い

私がBotというものを初めて知ったのはQuakeを手に入れたあと，

「な～んかないかな～」

と海外のftpサイトにあるQuake関連ファイルを漁っていたときでした。Botというディレクトリがあったので，そこにあったモジュールをダウンロードしてQuakeを起動，Botを発生させてみると，一見プレイヤーに見える方々がピョンピョン跳ね回っているじゃないですか。しかし，跳ねることに熱心であまり戦う意志が感じられなかったので，片っ端からロケットランチャーでボコボコにしてやりました。

それからしばらく経ったある日，また同じようにQuakeがらみのファイルを漁っていると新しいBotを見つけたのです。早速ダウンロードして試してみると，結構巧みに動き回ってるんですわ，Botちゃんが！　ちょうどQuakeの大会があるってんで練習しましたね～私は！　ガンガンBotを血祭りにあげて，

「いっや～，こんなに強すぎて上位総ナメにしちゃったら皆に怨まれちゃうかな～」

などとほかの参加者への心配で気疲れしてしまう私でした。

Quake大会当日，さらに電車内でイメージトレーニングを行った結果，あまりにも私が強すぎるためにほかの参加者からクレーム殺到，結局私がハンデとして両足での操作を強いられるという絵図が繰り広げられました（イメージトレーニングでもなんでもなくて妄想）。

「う～ん，足で操作するからにはやはりあらかじ

図2　跳ね回る人々……

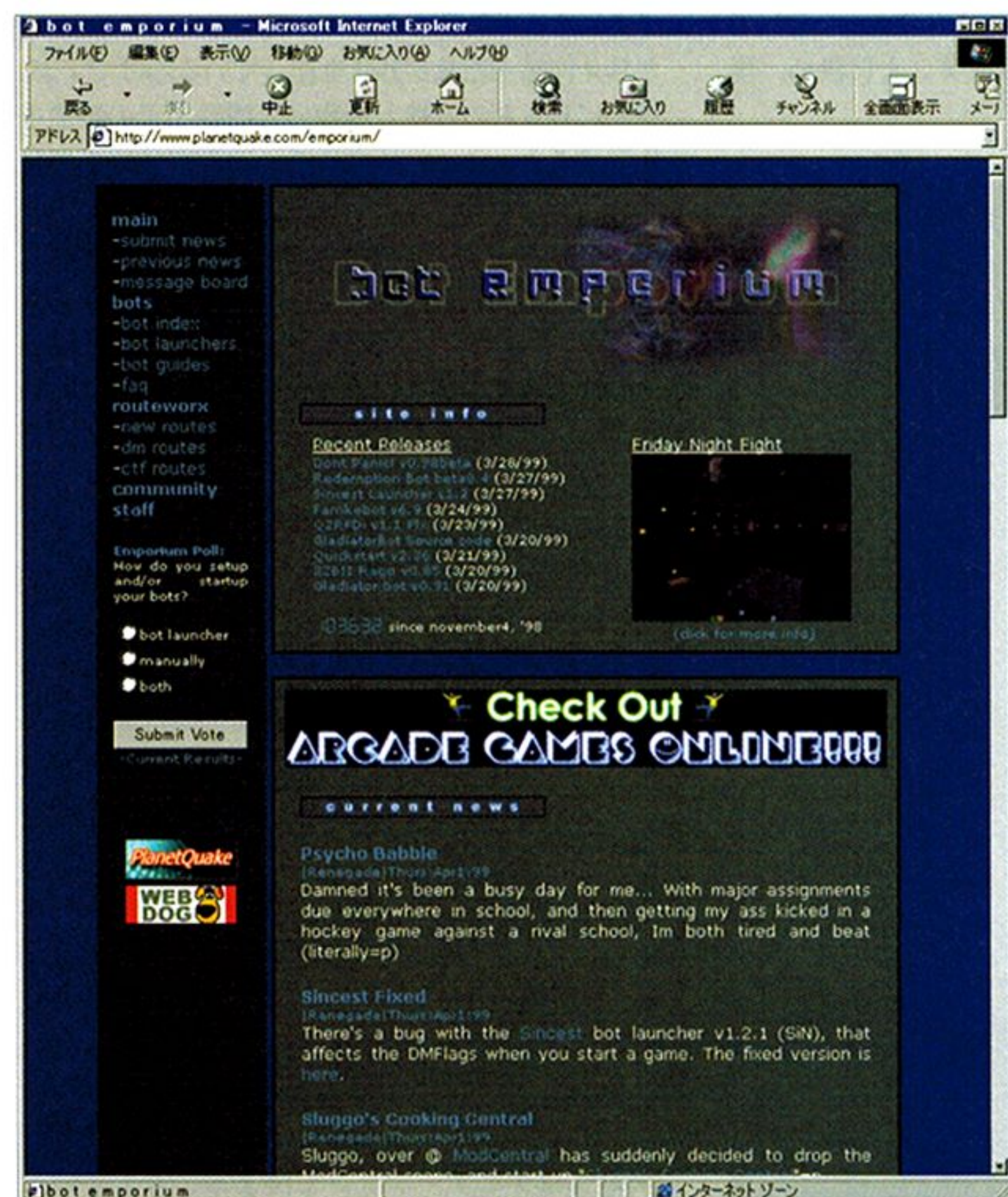



図3　Botプログラムを集めたWebページもある

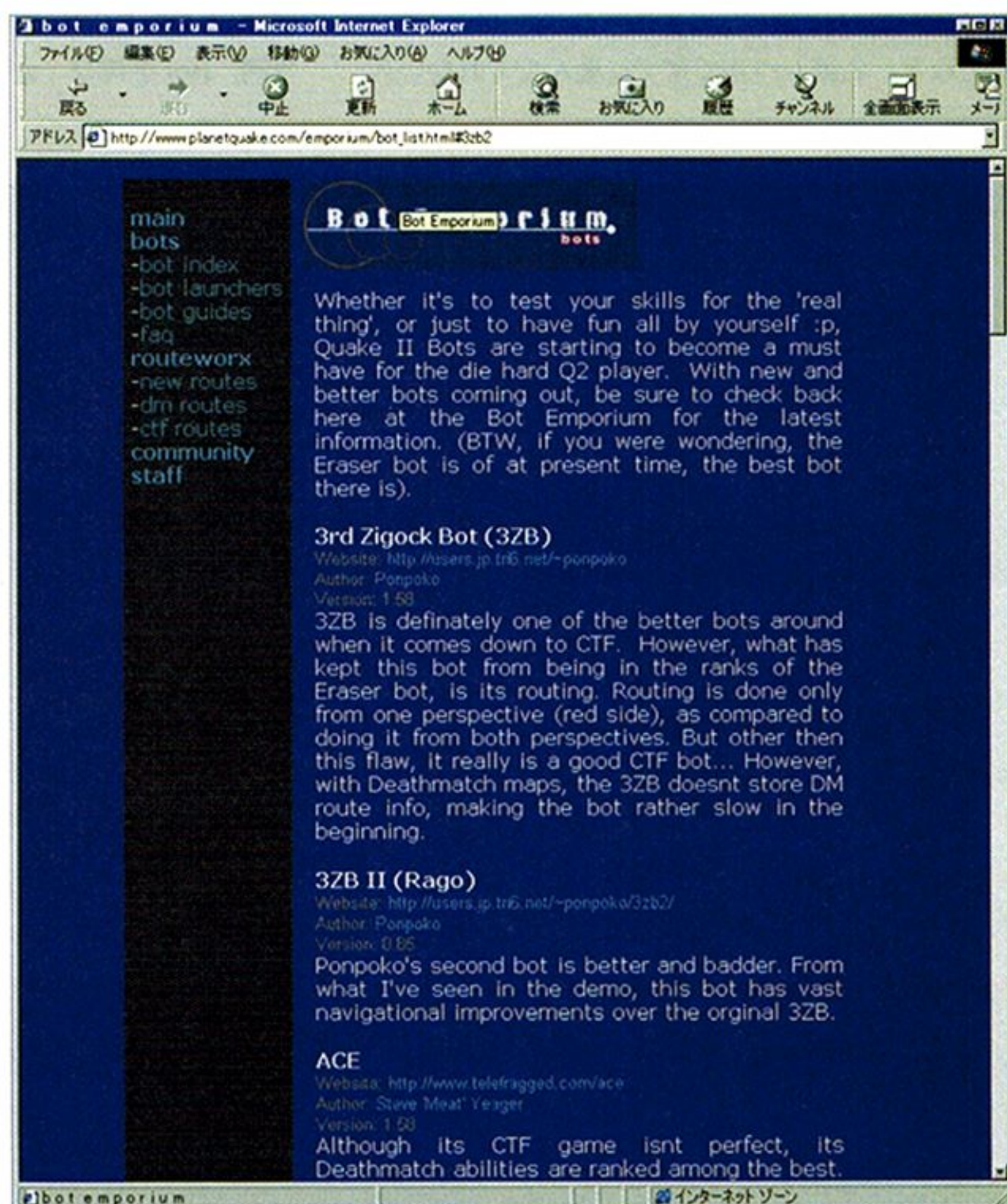



図4　3ZB，3ZBⅡの評価。ほめてるようなけなしてるような……

め駅のトイレで足を洗浄しておくのがエチケットであろうか？」

バカなことを考えながら会場へ到着，で，めでたく大会終了！　気になる戦果は～？　ビリやんけ！

確か記憶ではゲームを2つのリーグに分けて行ったのですが，1つめのリーグでビリだった私はお情けで2つめのリーグにも参加させてもらったにもかかわらずビリ！　ビリ！　私は怨みました，Botを。

「あんなんで練習したって勝てるかボケー！」

■いきなり失敗したBotの海外デビュー

私はこの1年間，プライベートな時間の半分近くをQuake2のBot作り一筋に……生きてきたかどうかは疑わしいですけれど，とりあえず作成に励んできました。

私が昨年初めてリリースした記念すべきBotの名前は「Stupid Bot」(バカ，まぬけ，つまんねえBot)！　この名前をひっさげて海外のBotシーンへ華々しく正々堂々のデビューをはたしたわけです(私ってウルトラバカですね)。

このBot，えらく評判が悪かったです。なにが悪かったかって，名前がえらく評判悪かったです(ちなみにほかのBot作成者はみんなカッコイイ名前を自分のBotにつけていました)。海外のBot関連のフォーラムでは，

「ひでえ名前だ」

「なんにしろ，こんな名前のBotとプレイしたいとは思わないね」

などなど，名前レベルですでに遊んでくれない人が続出状態。おまけにメールしたBotのバグレポートに対する私からの返事が，

「直せ，私，そのバグ，ジャンプについて，今週末，たぶん，これ，深刻，ありがとう」

という意味不明な英語ばかりなのでみんな気味悪がって引いてしまったという可能性もあります。

その後反省して，名前を3rdZigockBotと改め(前回のCD-ROMに入っていましたね)，そして今回，Botを再び一から作り直すことに決め，現在の3rdZigockBot IIリリースに至りました。このままの勢いで次発売のQuake3 ArenaでもBotを……と思ったら，最初からBotが入ってるんですって！

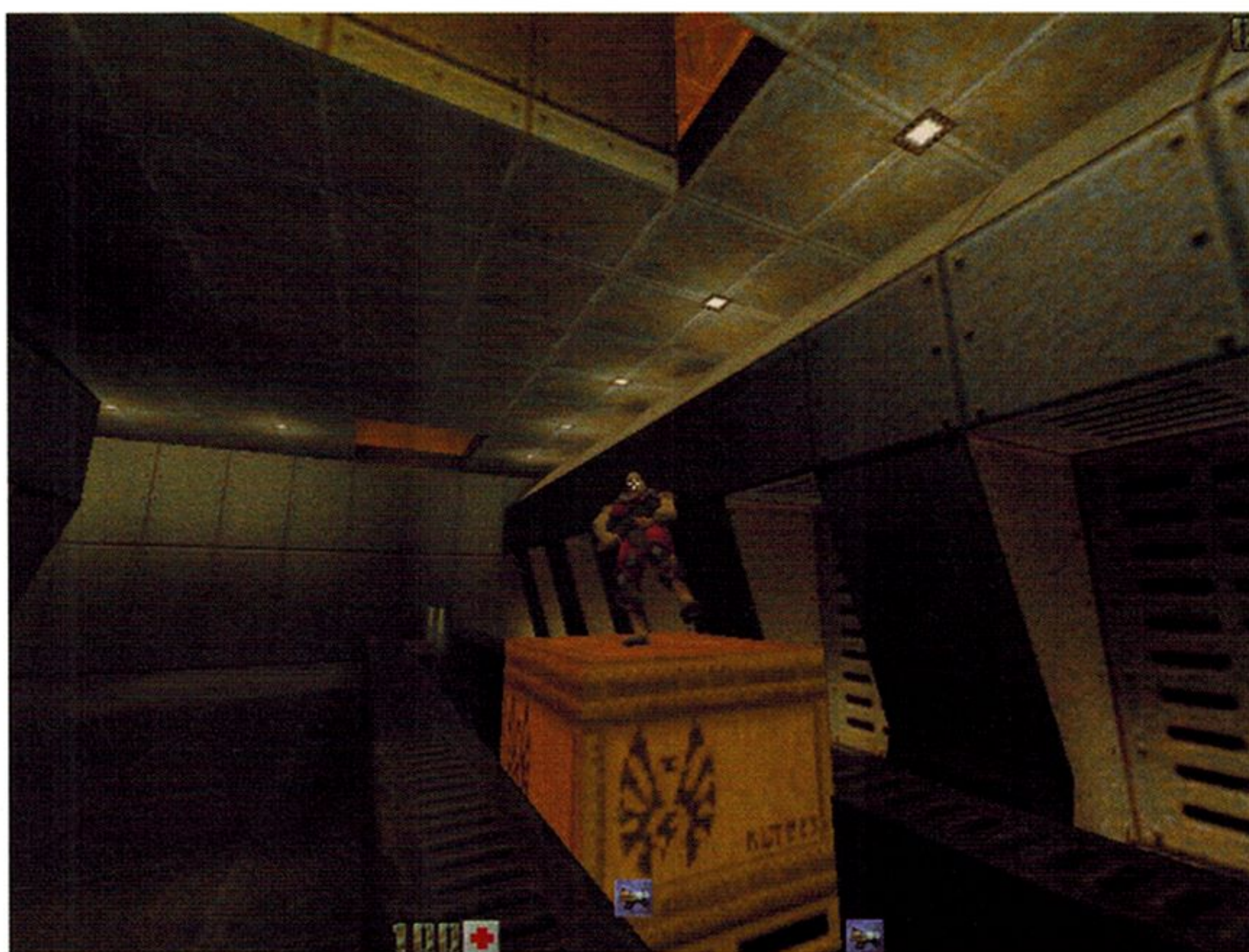



図5　障害物の上などに乗るのも，ルートファイルで指定してるから可能なこと

■Botを考える

「あんなんで練習したって，勝てるかボケーー！」とBotに怒った私がBotを作っているわけですが，確かにBotとの対戦と本物の人間様相手の対戦を比較すると，Botのリアリティは人間様相手の足元にも及ばないというのが現状です。

実際に人対人で対戦する場合，お互い本物の人間相手ですから負けられないという緊張感がありますし，負けたときは悔しさ，勝ったときは優越感が得られますよね。マップ上をその人の意志でウロついて，攻撃して，ジャンプしまくるのでアクションのバリエーションは無限個です。チャットで罵声を吐きまくればますますエキサイト。

一方，Botとの対戦はCPU相手なので勝っても負けても関係ありませんし，チャットでBotからの返事もなし(完全シカト状態)。アクションのバリエーションは有限個。相手に心理的変化がないので，人間相手のときに比べると，いまひとつエキサイトできません。しかし，これらの問題中で改善できるとしたら，ひたすらアクションのバリエーションを増やしていくぐらいしかありませんな。それともBotとのチャットを実現させるために学習型のAIでも作って言葉を1個1個教えていくのですか？　いやだーい！

とまあ，人間様にはかなわないBotですが，上記の件はまだ許容範囲です。私はBotの良し悪しの50％はナビゲーションで決まると考えています。Botがマップ上を縦横無尽に走り回ることが

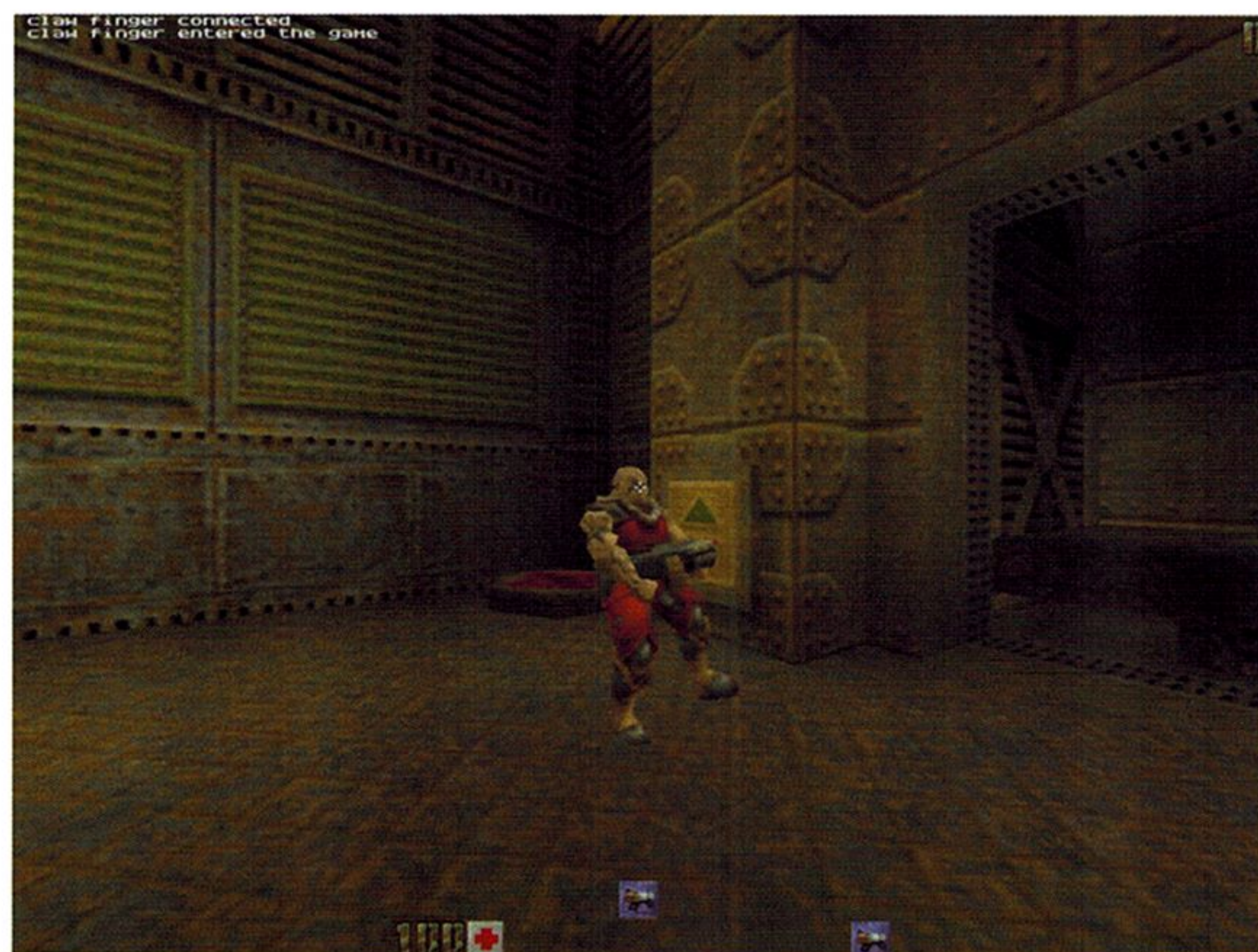



図6　歩き回るBot君

図7　ハデにやられてます。まだまだ人間様にはかなわないか

できなければ，その時点でそのBotはクソ確定であるとすら思えるのです。

現在，多くのBotは各マップごとに「ルートファイル」と呼ばれるBotをナビゲートするための道筋データを格納したファイルをあらかじめ作成しておき，ゲームの最中，Botがその情報を参照してマップ上をかけずり回るという手段を採用しています。もし，ルートファイルを使うことなく，人間の視覚並みに空間を認識させる処理(「これは階段だ～」とか「あそこをジャンプして回り込めばあのアイテムが取れる」などの判断を可能にする)をリアルタイムに実行しながらBotをかけずり回らせようとすると，死ぬほどCPUに負荷がかかることが容易に予想できます。こんなBotだと4体程度出現させただけでゲームが止まってしまうんじゃないですか？　しかし，今後，高クロック動作の怪物CPUが出てくれば可能かもしれませんね(でも私が思うに，たぶんムリ)。

今年idSoftware社からリリース予定となっているQuake3 ArenaはBotによるCPUへの負荷は相当であるようなことを最初から宣言しています。プロが作って負荷が相当というのですからきっとスゴいBotなんでしょう。でも，ひょっとしたら上記のルートファイルを使わない方法でBotのナビゲーションを実現しようとしているのでしょうか？　……危険だ。

メーカー自らサポートするほどメジャーになってしまったBot(UnrealにもBotが入っています)。このままだと一般ユーザー作成のBotの出る幕がなくなってしまうのではないかと少々心配ではあります。しかし，ユーザーカスタマイズを前提としたソフトウェア設計を行い，ソースファイルなどのSDKを積極的にユーザーへ提供するメーカーも増加しつつあるので，チャンスはあります(Botに限らずカスタマイズしたモジュールを作ること全般において)。

面白い例としては大学のゼミで人工知能の研究のためにQuake2のBotを作成してみたなんてのもあります。

「先生，ボクの卒論テーマは“QuakeのBot”でいかせていただきたいと思うのですが」
と提案してみてはどうでしょうか？　おそらく，
「キミは卒論をナメているのですか」
と却下されると思いますけど。

■これからのBot人生

これからあと何年Botを作るかわかりませんが，とりあえず今はQuake2のBot作成を続行しながらHalf-LifeのSDKの一般公開時期をうかがっている次第です。

こういったSDKがリリースされるや否やアメリカあたりだと即あちらこちらに「～Project」というホームページが立ち上がり，皆せっせと改造モジュール作りを始めます（現時点でSDKリリース前からすでにツールを作成して待ってる奴までいます。リリースされなかったらど～すんの？)。

Quake，Quake2ともにタイミングを逃してしまった人はSin（もうソースコードがリリースされています）かHalf-Life，Quake3あたりを狙ってみてください。

ガッツ石松氏の名言「政治やりてえな」に感銘を受けた私は今年も「3Dネットワークゲーム改造モジュール作りてえな」でいきますぞ。

私事で誠に恐縮ですが，誰か！　このモデム買い取ってク・レ・ヨ！

図8　なんか変なのがいる……

特集 ネットワークゲームの地平へ

第二の人生, Ultima Online

Kazuhisa@Pacific

RPG好きな人なら誰もが一度は夢見るようなバーチャルワールド。いち早くそれを実現し，いまだにネットワークゲーム界に君臨するのがUOことUltima Onlineです。古参UOプレイヤーが見たUOの変遷からネットワークゲームのあり方を探っていきましょう。

Ultima Online(UO)の説明はいまさら必要ないと思う。米Origin Systems Inc.(OSI)の古きRPGシリーズ「Ultima」の最新バージョンにして，3000人が同時にひとつの世界で遊べるという，サーバ依存型ネットワークRPGの大御所だ。日本国内でも，発売当日にT·ZONEに行列ができたりしたので，ゲームの中身がどんなものかは知らなくても名前ぐらいは知っている人も多いだろう。

この手のネットワークRPGの真の先駆者は「Meridian59」[*1]だと思うが，シリーズを何作も重ねているにもかかわらず，商業的に成功したとはいえないように見える。しかし発売から2年と5か月が経過したUOはいまだ健在で，飽くことなき機能追加/修正，バグフィックスなどが行われている。筆者自身はβテストからの参加となるが，お金にシビアなアメリカ人(遊んでいようがいまいが月に9ドル95セントの課金があるのだ)とネットワークインフラ(電話代)が高い日本で，これほどまでに流行ろうとは思わなかった。

さらに。いまでこそ日本語でのチャットもできるようになったが当時は英語必須ともいえる状況で，「世界で唯一英語ができない国民」である日本人が抵抗なく入ってきたのも意外だった。

UOの魔力は偉大だ。テレホにもかかわらず電話代が12万円ぐらいになったときや，夢の中で鳥を狩って羽根をもいでいる自分に気づいたときはかなりヤバいと思ったが[*2]，まぁそれほどまでに危険なソフトなのだ。かつて某誌に「親の遺言級」という言葉があったが，まさにそれに値する逸品だといえるだろう。寝食を忘れ，トイレも行かず，会社も忘れ，学校も忘れ……。

以下は，あくまでもすべて筆者の主観だが，UOについて徒然なるままに書き綴ってみた。思い出，苦情，文句……頭の悪い古いプレイヤーのたわ言なのかもしれないし，要するに「UOってどんどんつまんなくなってないかい？」といってるだけなのだが，プレイヤーも，プレイヤー予備軍も，読んでもらえれば幸いだ。

＊1　The 3DO Companyが発売したネットワークRPG。チャキチャキとバージョンを変えているが，UOほどの人口は集められない模様。結構挙動も軽くて面白いのだが……。筆者も一時期プレイしていたが，なにかが，こう，欠けている気がする。

＊2　12万円の話はホント。かなりびっくらこいた。ちなみに夢のときは，起きて最初に思ったことは「あの羽根，どこにしまったっけ？」だった。重症。救いようありません。

■進化と退化を繰り返す「世界」

まず最初にいっておきたいが，とにかくにもUOは偉大だ。回線状況に左右されやすい(インフラ上の問題)とかゲームの内容があまりにも頻雑に変更されすぎる(ゲームシステム上の問題)とか，サポートが悪いとかGM(Game Master)が仕事しないとか，なんだかんだとさまざまなことをいわれているが，いまだに世界で20万人以上のプレイヤーを抱える巨大なネットワークRPGであり，3月半ばの原稿執筆時には，まだその偉業をほかのどこもがなしえてないのだ[*3]。

製品発売からしばらくの間はバグ(というかシステム上の穴，というべきか)も多く，OSIも試行錯誤でシステム変更を行っていた。いまにして思うに，あのころのOSIは「Role(役割)をPlay(演じる)する」というキレイごとの部分と，約10ドルとはいえ毎月お金を取っているがための「ビジネス」の部分をうまいこと両立させようと悩んでいたのではないだろうか。RolePlayとしてはどう考えたってヘンなパッチ[*4]を当ててみたり，その翌週には戻してみたり，プレイヤーともどもOSIも模索していたに違いない。

パッチ当たるとなにが変わるか？　武器の強さが変わり，モノの価格が変わり，モンスターの強さが変わり，「してはいけないこと/いいこと」が変わり……いうならばゲームシステム全体に変更が及ぼされるわけだ。昔はずいぶん小刻みに変更が行われて，プレイヤーはやたら混乱させられたものだ。それについていけなくなって去っていった者も多い。

＊3　といっているうちに，いましがたEverQuestが届いた(3月23日)。米Sony Interactiveが放つネットワークRPGの大作にして真打ち。今年登場する(であろう)ネットワークRPGのなかでも期待の高い一作。EverQuestか，Diabloか，Asheron's Callか，Revenantか。世界レベルで20万人のプレイヤーを集めるかどうかはわからないが，むろん筆者もプレイする。UOからの移籍組も多いので，UOのサーバはちょっと空くかもしれない。ラッキー。

＊4　機能変更や修正などを行うファイル。更新されると自動的にダウンロードしてクライアント側に当ててくれる。ダイヤルアップ環境で数MBものパッチを当てるのだけは勘弁してほしい今日この頃。

■老人のさまざまな思い出

何十回ものパッチを繰り返し，最近はずいぶん様子が変わった。β初期のころから見ている人間ならわかるかもしれないが，まるで別世界のようだ。

その最大の変化は，まずもってNewbie[*5]に限らずプレイヤーに優しくなったということだ。発足当時が嘘のようだ。お金はアッサリと稼げるし，

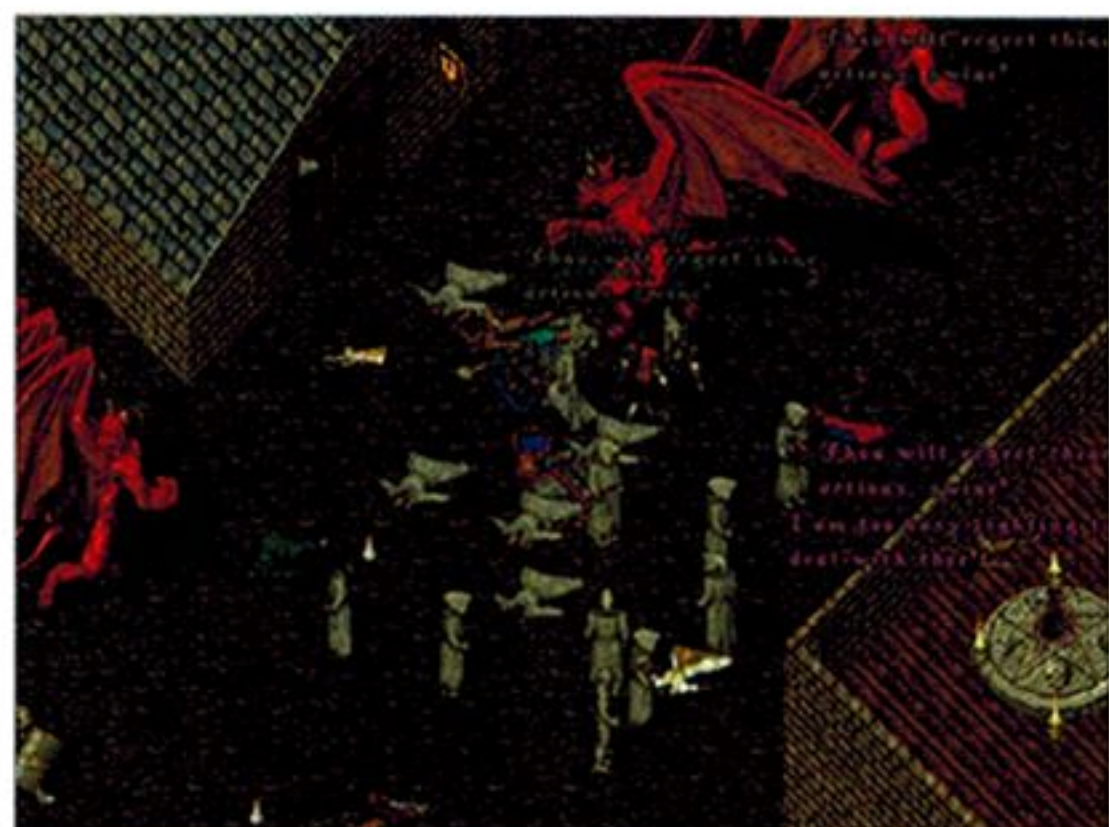



図1　今となっては伝説の，UOのβテスト最終日「Britannia滅亡の日」の様子。世界中の街にデーモンがなだれ込み……衛兵も店員もプレイヤーも，全てが殺されて世界は終わった

図2　死んだあとで，最後だからと思ってGhostのままLord British城内に潜入……したが，デーモンで埋め尽くされて，白黒の世界でしか城内を拝むことが許されなかった

図3　αテストに近い段階での画面ショット。どっから見てもUOだが，かなり違う雰囲気のものになっている。ちなみに転がっているモンスターはOrc

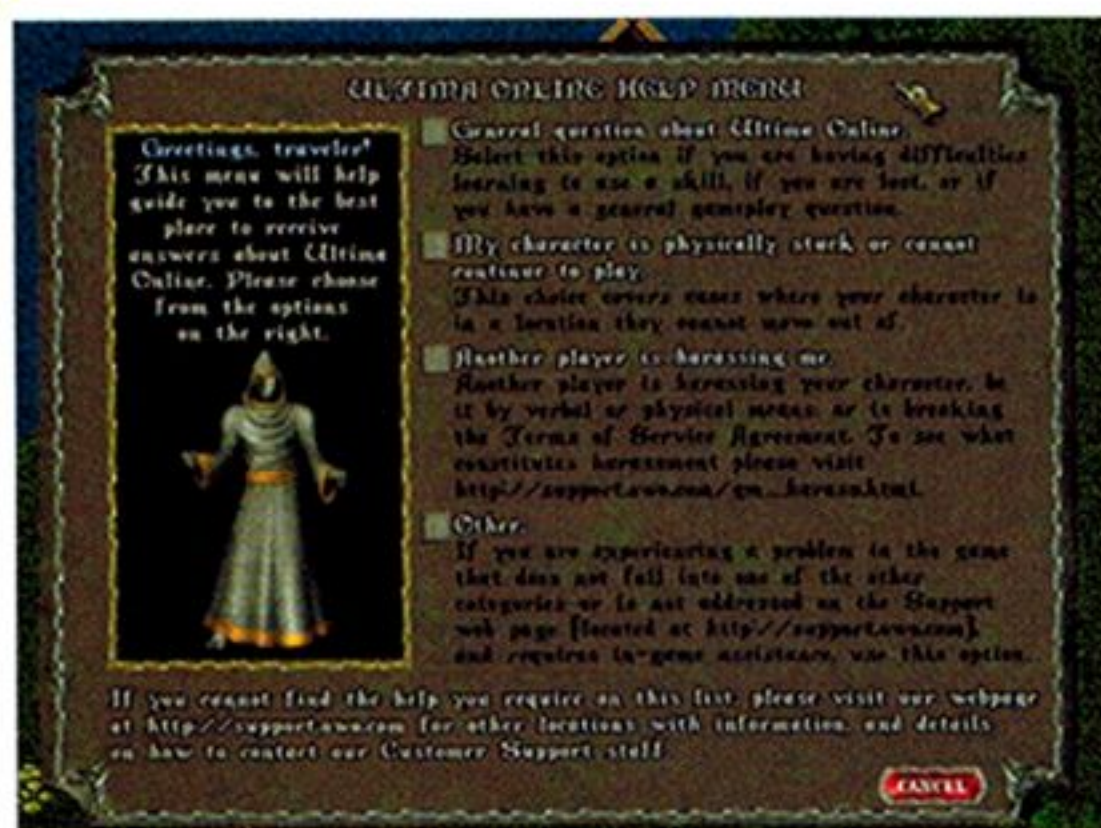



図4　先ごろ変更されたHelpメニュー。GMにはかなりの権限が与えられているので，頼んだことをアッサリとやってのける人や「俺にはできない」といい張る人，立ち話をしだす人など，いろいろといて面白い。でも人の家の中でくつろぐのはやめてくれ（笑）

図5-1，2　いま現在のネットワークRPGゲーマーの期待を一身に背負っているであろうEverQuest。どこまでUOの牙城を崩せるかが見ものだ

図6　Meridian59のシステムでEverQuestをやっているかのようなAsheron's Call。実は個人的に結構期待しているものだったりする。3月26日からβテスターを募集している

図7-1，2　当初の予定が延びに延びて，もうどうなっちゃってんのかわからなくなってるDiablo2。しかし，どれほど遅れても今年（かどうか知らんが）イチ押しであることに変わりはない。筆者は何度となく動いているのを見て実際に触ったが，やはりデキはいい。ま，秋までに出ることを期待しつつ……もうバランス取りのレベルまでは完成しているらしいし

図7-3　Eidos InteractiveのRevenantも，ダークホースではあるがイケそう。一部では「Dia2」を超えた，とまで囁かれている逸品。出てもいないものを超えるなよ。しかし，国内でアイドス・インタラクティブが完全日本語化して英語版と同時出荷のようなので，ある意味いちばん期待できる作品ではある

Newbieにしても先人の知恵をたっぷり仕入れているので，やたらに効率よく生活基盤を築いていく。我々初回組が右も左もわからずにあほうのようにさ迷って，あほうのようにダマされて，あほうのように殺されていたころからは考えられない。

●

Kindling（焚き付け）を拾って1gpで道具屋に売ってシコシコとお金をためて，魔法のScrollを全部揃えたときには感無量だった。やっといっぱしのMageになれたと思ったものだ。何度もだまされ，何度も殺され，何度も死んで，ようやっと集まったSpellbook。Tailor（仕立て屋）でCap（帽子）を作ると異様に儲かると知ったときには，小躍りしたものだ。中級者と呼べるようになったときに，初めて2人分のお金を足して小さな家を買い，海岸線沿いに建てたときの喜びは，もう遠い昔の出来事になってしまった（といいながら，現在でもそこに住んでたりするのだが）。

そこらの知らない外人10人ぐらいに「Hey, Kaz. ドラゴン倒しに行ってみないか？」と誘われて，いっぱしのMageでいたつもりだったのでついて行ってほんの数秒で全滅したことも。離れ小島で全滅して電話で友達に助けを求めたことも（笑）。Loot[*6]されてそのプレイヤーを執拗に追いかけたことも。PK[*7]とお友達になっていろんなアイテムをもらったことも。海岸で釣りしてるところを魔法で殺されて根こそぎアイテムを奪われたことも。初めてElementalを倒したときの喜びも。手強いモンスターが怖くて，いつも3，4人で行動していたことも。「PKされてアイテム盗られた！取り返しに行くからついてきてくれ！」といわれてついて行ったら，すっかりダマされてPKの城だったことも。友達が戦っているモンスターに魔法を撃とうとして間違えて友達に当てて，すっかり称号を下げたことも[*8]。GreatLordになって初めてOrder Shieldをもらったときのことも。ダンジョンで出会った外人に気に入られて，家に招待されたことも。Orc砦にパーティ組んで行って，PKの集団に出会って全滅してすべてのアイテムを取られたことも……。

当時はムカついたり喜んだり悔しがったりしたものだが，すっかり遠い思い出になってしまった。なんだかとっても懐かしい。

＊5　いわゆる初心者。「i'm newbie!　Give me spare stuff, plz!」と何万回聞いたことか。

＊6　自分の死体からアイテムを持ち去られること。「死んでるんだからLootされて当然」だの「アイテムは守ってあげるべきだ」だの，いい悪いの論争は昔からあるが。そんなことはどーでもいい。Lootするのもされるのもロールプレイ。他人の生き様に文句つけないように。

＊7　PK。Player KillerまたはPlayer Killを指す。要するに「プレイヤーを殺すプレイヤー」ってなわけだ。これまた存在の是非の論争が絶えないが，そんなことはやっぱりどーでもいい。ひとつの「世界」なんだから，悪い奴がいたっていいじゃない。集団で襲うのは当然だし，悪い奴なんだから初心者だろうが知り合いだろうが殺す。いいじゃない。悪い奴がいない世界ってのもねぇ。しかしまぁ海外でもさんざん論争のネタになったらしく，事実上PKを許さないゲームも多い。つまんねー。まぁ軟弱な日本人にはちょうどいいか。

＊8　Lightning（電撃）の魔法は8レベルあるうちの5番めだが，昔は異様に強かった。モンスターハンティングに便利だが，間違えるとそういう目にあったものだ。ちなみに筆者は，自分に当てて死んだこともある大馬鹿者。

■RolePlayと バランスの微妙な問題

それがいまでは，作って2日めぐらいのキャラクターでもフルSpellbookを持ってたりする。1週間もあれば家を買ってVendorを置いていたりするし，初心者のくせにダンジョンに行きたがる。効率よくお金を稼ぎ，効率よくSkillを上げ，効率よく称号を上げていく。GM Swordsmanになり，Glorious Lordになり，家を買い，Vendorを置き，Guildを作り，そしてまた新たなキャラクターを作る。あぁ，もちろん全員がそうだというわけではない。そういう人も多い，という話だ。

図8　我が家。2階建てやTowerのDeedも持ってるのだが，建てる場所もないしアイテム移動が面倒だしで，いまだここに住んでいる。まぁ別に小さい家で困ることなどないのだが

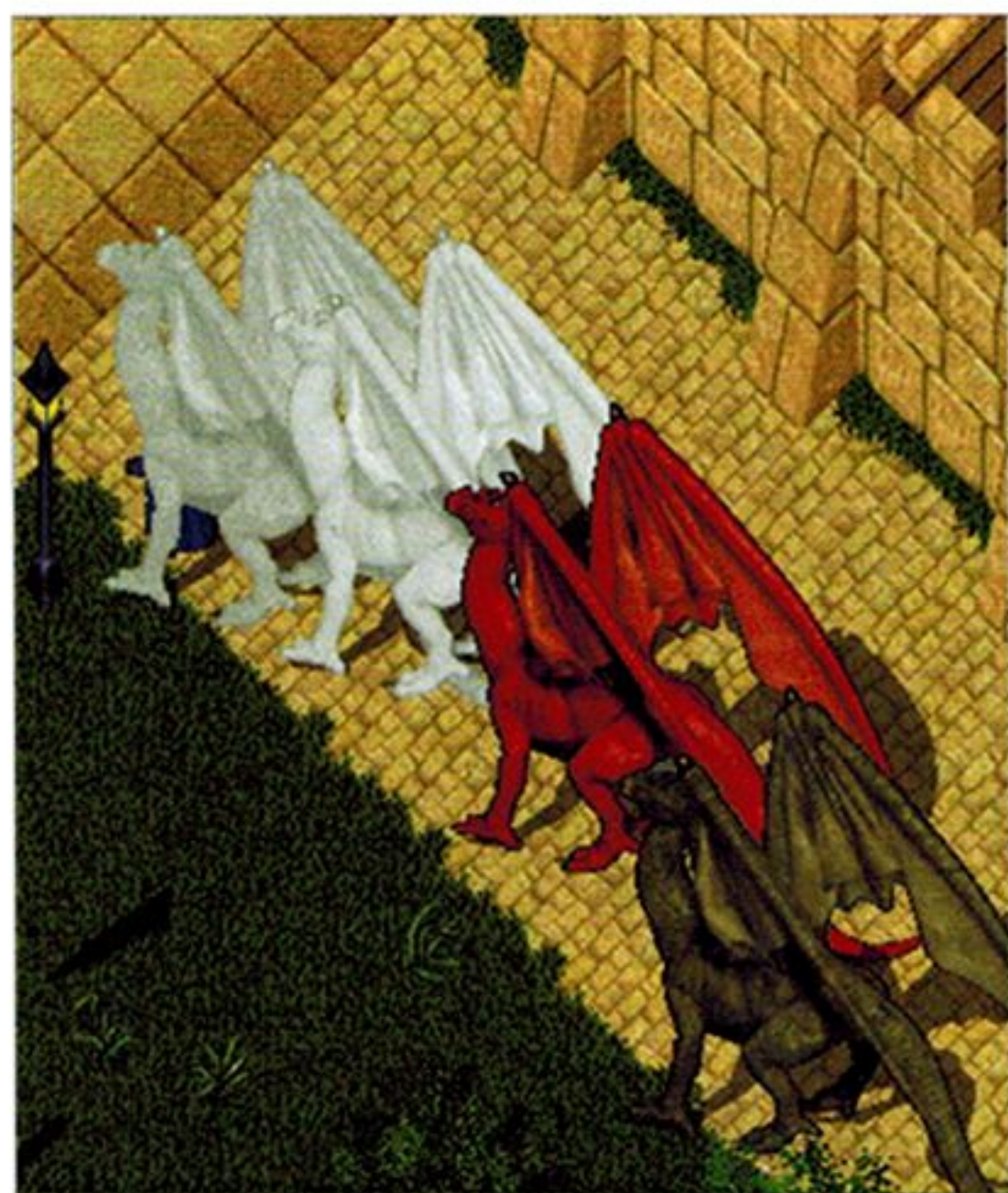



図9　DragonとNightmareという2大モンスターを仲間にできる唯一の職業がAnimal Tamer（調教師）。こうやって4匹並べて行進するのも，険しい道を乗り越えてきたMaster Tamerの特権だ

図10　Ghostから復活して，自分達の荷物を取りに行く図。当然あるはずなどないのだが，この頃はまだアイテムが惜しい時期だった。最初に悔しかった大きな出来事はこれかなぁ……製品版開始から2週間めぐらいのこと

なんていうか……そこには，自己の分身であるキャラクターを育てる余裕がない。ただひたすら数値にこだわり，PowerPlayをしているだけだ。もちろん「数値にこだわるな」というのがキレイごとなのはわかってる。強くなければ生き残れないし，ダンジョンに行けない。強いモンスターを倒せないし，いいアイテムも手に入らない。でも，本来UOが目指したものは，そこにはない（と思う）。

昔のプレイヤーは，もっとRolePlayを楽しんでいた。電子の世界「Britannia」で繰り広げられるさまざまなイベントを「プレイヤー」として楽しむ余裕があったように思う。「俺は魔法は使わない。仲間もいない。正々堂々と剣で勝負しろ」といってDeceitの中で戦いを挑んできたPKや，「仇はとった。奴の首だぜ」といってVesperのBank前にゴロンと首を捨てたPKK[*9]を見て，仮想世界とわかっていながらもしびれたものだ。

ダンジョンに行って死ぬのもよし。腕試しでデーモンに挑戦して瞬殺されるのもよし。お金ためて買ったDeed[*10]をThiefに盗まれるのも，人助けをしてダマされるのも，ネットワークRPGならではのイベントだ。せっかくかつてない世界の住人なのだから，もっと余裕を持って楽しんでもいいのでは。

あぁ，なんだかいいたいことがうまくいえない。要は「いまのプレイヤーはだいぶ甘やかされているけど，なんか違うのでは？」ということだ。もちろん，それが良いとか悪いとかいう話をしているのではない。僕のような「古い」プレイヤーも，最近始めたばかりの「New」プレイヤーも，すべて飲み込んでしまう世界，それがUOだ。

●

数々のパッチを経て，もっとも変動があったのはSkill関連だ。職業がずいぶんと細分化されてきた。

ハイレベルなSwordsmanがハイレベルなMageを兼任（？）することなどRolePlayではちゃんちゃらおかしいことだし（少なくとも僕はそう思う），ダンジョンの宝箱に罠がかかっていないのもヌルすぎると思う。デーモンクラスのモンスターをひとりで倒せるプレイヤーがいるというのもどうにも納得いかない。普通のRPGではありえないことだ。OSIは「役割分担を決めてパーティでHuntingに行ってほしい。みんなで遊んでこそのネットワークRPG」と思っているのだろうし，事実Britanniaの世界はそのようにシフトしている。

MageやBardは壁となるSwordsmanがいないと生き残りは難しいし，Swordsmanはほとんど魔法は使えないような状態なので，Mageなどの補助がないと強敵に立ち向かうにはツライだろう。ダンジョンではRogueの「罠解除」の能力がないと迂闊にドアさえ開けられない。Treasure Hunting（宝掘り）では，地図起こしや穴掘り，罠解除などの能力がないと掘り起こすことは不可能だ。

それとは逆に，追加能力（？）をどんどん付加されて脚光を浴びる職業もある。

宝の地図を釣り上げられる「Fisherman」，作れるものが大幅に増えた「Tinker」，Skill値が100になると銘入りの製品を作れるようになった「Blacksmith」「Tailor」，作れるPotion（薬）の種類が山のように増えた「Alchemist」[*11]，禁断の魔法を唱えられるようになった「Mage」[*12]，いまやありとあらゆる家具を作れるようになったといっても過言ではない「Carpenter」。当初の予定なのか，ユーザーからの要望なのか，それとも単なる思いつきなのか，OSIは次々と新機能を盛り込んで送り込んでくる。

●

昔のように「ひとりでなんでもできるぜ！」というキャラクターを作ることは事実上不可能になったが，これはこれで歓迎すべき方向性だろう。「俺は戦闘なんてなーんもできないけど，武器作らせたらスゴイよ」とか「Orcが相手でも肉弾戦だと死んじゃうけど，魔法なら黒デーモン[*13]でもなんとかなるよ」とか，そうなればパーティを組んで行かざるをえなくなり，もっと知らない人と接する機会が増えるかもしれない。街でパーティを組んでダンジョン行ったら裏切られて即死，とか。そういうのもありそうで楽しい。

＊9　PKK＝Player Killer Killer。前出「PK」を殺すプレイヤーのことだ。要するに「いい奴」を演じているプレイヤー。とってもナイス。

＊10　家やVendor（店員）を置くための権利書。権利書をダブルクリックして使うことで，初めて設置できるわけだ。ウン十万ゴールドものDeedを盗まれると，本物の人生までやめたくなるらしい（経験者語る）。最近は家のDeedの買い戻しもやってくれるようになったので，Deed Thiefも増える……かもしれない。少なくとも筆者ならそうする。

＊11　原稿執筆時の3月23日時点では，まだこの機能追加はなされていない。苦労の割にはあまり報われなかったAlchemistが，一躍脚光を浴びることになるだろう。それにしてもOSI，一気に数十種類も増やしちゃって，やること相変わらず極端すぎ。

＊12　＊11に同じ。まだ反映されていない。そんなにいっぱい機能追加しなくても……まぁいいんだけど。

＊13　UO内最強のモンスターのひとり（？）。筆者は基礎体力のないMage/Bardなので，こないだ触れただけで死んじゃいました。アーメン。

■RolePlay……って？

発売されて，早2年5か月。その間にさまざまな状況が変化した。昔のような「世界はあげよう。あとは勝手にやってね」という状況はプレイヤーたちによって改善され，OSI自体も態度を改めてきている。筆者が生活を犠牲にしてNTTに膨大な額の料金を払って，しまいには夢の中にまで登場し，それでもなおプレイし続けていたあの頃のUOとは違う。夢の中で鳥を狩ったときには，正直いって本当にどっちがRealLifeなのか混乱した。やたらに厳しく，やたらに重く（笑），やたらにいろんな人がいたUOは，もはや過去の遺物なのだ。

むろん，自分自身も変わった。正直いって昔ほどの熱意を持ってプレイしているわけではなく，最初に作ったキャラクター，Mageの「Kazuhisa」でフラフラと生活している。お金もある。家もある。珍しいアイテムもたくさんある。ダンジョンに行っては人助けをし，街中でNewbie（ではな

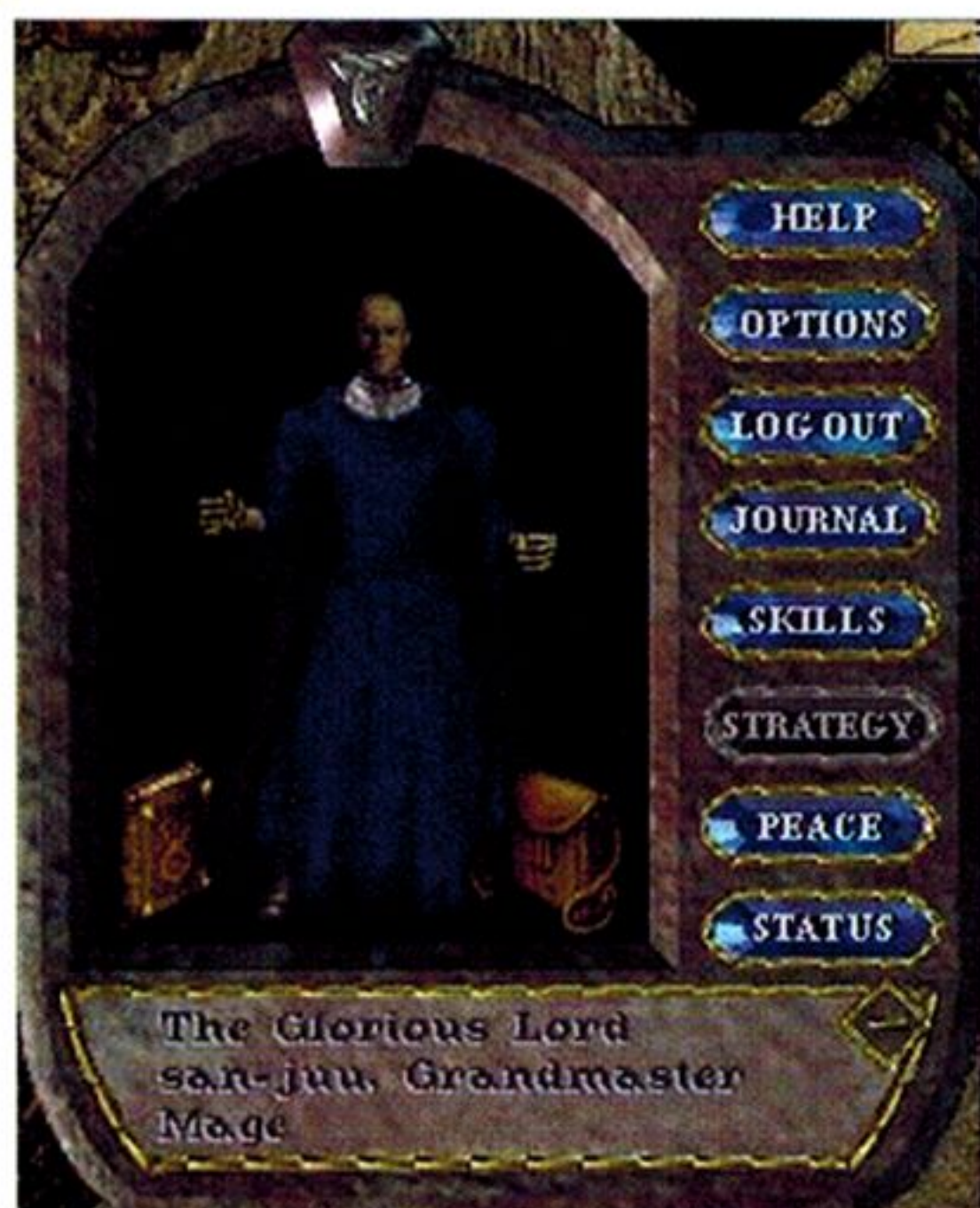



図11-1　最高の称号である「Glorious Lord」。大変な努力家か大変効率のいいプレイヤーかどちらかだろう。このあたりになると，自分の職業まで見せびらかすことができるので，通なら「Grandmaster Warrior」以外をキープすべし

図11-2　7つのSkillがGM（Grandmaster）クラスのキャラクター。考えうる最高の状態だろうが，個人的にはなんか違う気がする……ま，確かに数値を上げるのもRolePlayかもしれないが

い人もいっぱいいるのだろうが）に施しをし，そこらで見かけたGhostと立ち話をして助けたりして。それはそれで結構面白い。

PKには過敏に反応し，すぐにCheater*14呼ばわりした挙句「PKがいないshard*15を」といってみたり。殺されて文句いう，盗まれて文句いう。なんていうか……RoleをPlayする余裕なんてない。最近はPKのほうがよっぽどマトモではないかと思ってしまう。UOは，もっともっとFun To Playなものではないだろうか。PowerPlay結構，Thief結構，PK結構，FPK結構*16。でもなにか，最近のUOにはすごく大事なものが欠落している気がする。筆者の思っていたUOの世界は，もっと厳しく，もっと楽しいものだった。プレイヤーの質が落ちたのか，それとも筆者が古い人間でついていけないのか，それはわからないが。

＊14　Cheat＝インチキ，をしてる人。なにをもって「Cheat」と呼ぶかどうかは論争のタネだが，それもまたどうでもいい。ツールでインチキしてるのは論外だが，システムの穴を突っついているのぐらいはいいのでは。OSIがそのうち直すだろうし，直さないものは「仕様」ってことでしょう。

＊15　もともとは「破片」という意味だが，UOの世界ではサーバのこと。いつのころからかいい出して，すっかり定着した。なんとなくカッコイイ表現なので，筆者は気に入っている。

図12　各自家にVendor（店員）を置いて，アイテムを販売して商売することもできる。これはこれで需要を見越してモノ作ったりして，売れ行きがいいと結構面白かったりする。友達のVendorも置けるので，Vendorごとに役割を決めて，一大ショッピングセンターなんかも作れるわけだ

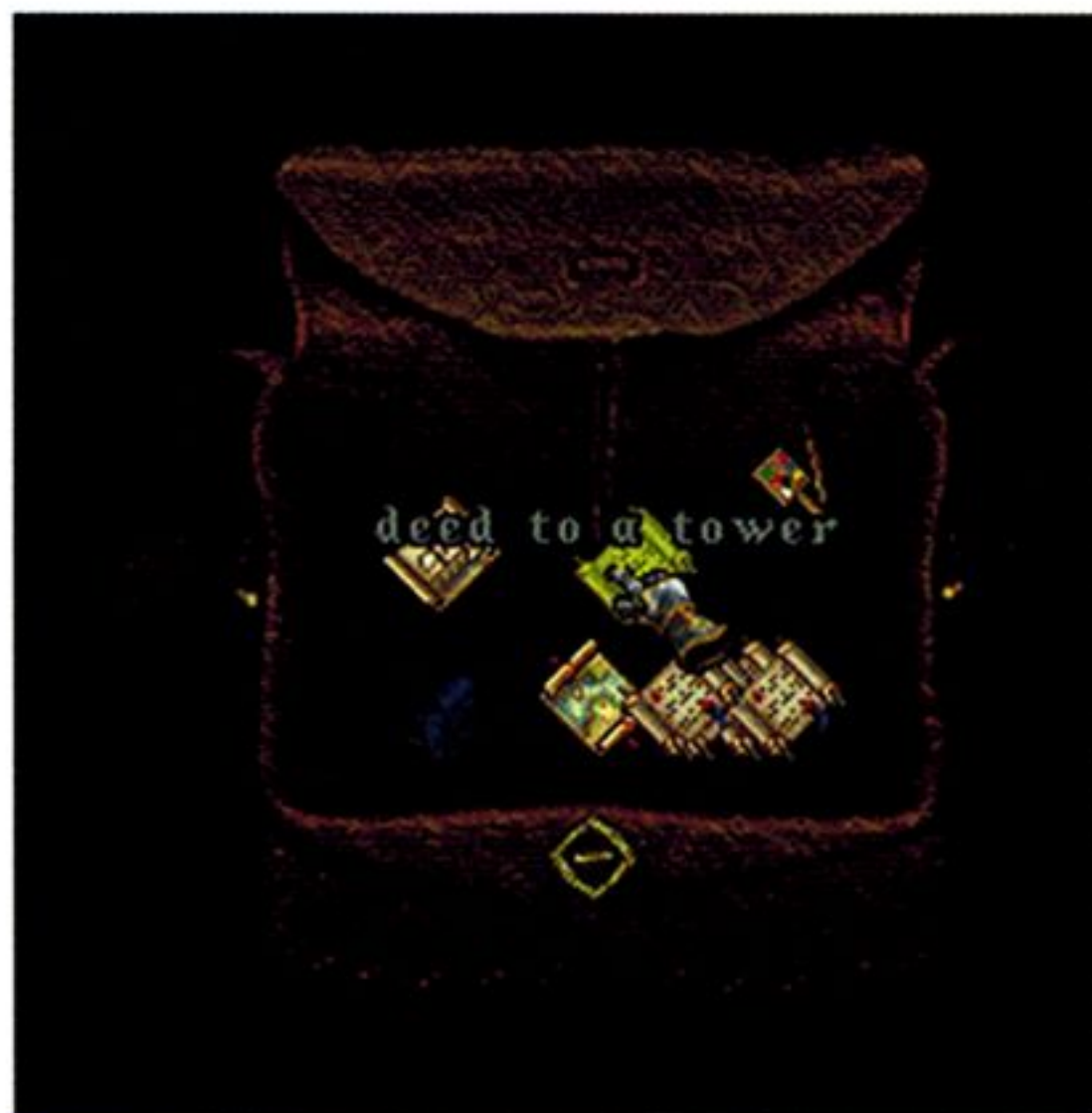



図13　盗まれると人生イヤになるTower Deed。盗まれた人の気持ちを察するとこっちまで気落ちしてしまう……

＊16　FPK＝Frag PK。システムの穴ともいえる個所を突いてきてPK行為を行う連中。どうにも卑怯な連中だが，まぁUOプレイヤーとしては防衛策ぐらい知っておくべきだろう。相手に「犯罪者フラグ」をつけるようなことをして（または強制して）サクッと殺すことなどが得意（こうすれば自分には「殺しをした」というフラグが立たないわけだ）。

■日本人キライ

話題はそれるが。

語弊があるかもしれないが，筆者は日本人UOプレイヤーが基本的にキライだ。日本人にはマナー悪い人が多いような気がする。二言めには「俺の獲物だよ」とか「なんかくれよ」とか「俺がRes*17しようと思ってたのに！」とか。なんだかなぁ，もう。むろん，そうでない人もいっぱいいるし，キャラクターの善悪を問わず素晴らしいプレイヤーもいっぱいいるが，そういう手合いは日本人で

図14　宝の地図は，Cartography（地図作成）のSkillを持っている者が解読する。解読できたら画面のように場所を示してくれるので，そこへ行ってザクザク掘るべし掘るべし。お宝を守っているモンスターをひとしきり退治すれば，ウハウハなお宝が手に入る

多く見かける……気がするのは筆者だけだろうか。

パッチ情報を読まないで騒ぐ，ダンジョンでPKにやられて騒ぐ，英語で話しかけられて「nani itteruka wakanne-yo」といって逃げる，すぐに群れて行動したがる。

ドラクエじゃないしFFでもない。もともとハードなゲームなのに，完全に勘違いしている節がある。群れてみんなでSkillUpするのは結構だが，パッチ情報ぐらい読みなさい。PKぐらいで騒ぐな。英語ぐらい話せ。そんな連中が次の新規プレイヤーにまた教えを施して，どうしようもないプレイヤーを大量生産してるだけ。「日本人はマナーが悪くて不勉強でウザいので，日本サーバをいっぱい作ってそこに放り込んでおきたい」からいくつもの日本shardが設置された，という噂も一瞬信じてしまいそうになる。

無論「我こそが素晴らしいUOプレイヤー」というつもりは毛頭ない。しかし，少なくともそういう手合いよりはUOを楽しんでいる自負があるし，せっかくのネットワークRPGなのに日本人で群れるという愚かなことをする気もない。日本にあるからといって「日本人サーバ」ではない，ということぐらいわかってるつもりだ。米国産のネットワークRPGなんだから，英語ぐらい話すのは義務だろ。「ちょっとぐらい話せるからってエラそうにいうな」というのもわかるが，話せないならUOをやる資格はない*18。

＊17　Ressurectionという第8サークル（最も難易度が高い）呪文で，死人を生き返らせることができる。呪文詠唱のときに「An Corp」という文字が見えるので，その文字を見ると「あぁいい奴なんだな」と思ってしまう。難易度高いので，呪文のSkillUpがてらやりたがる人も多い。

＊18　まぁなんのかんのと意見はあると思うけど，やっぱりUO内では英語でしょ。少なくとも，英語で話しかけられて逃げたり日本語で返したりすることだけはやめてほしい。あー恥ずかし……っていうか「UOプレイヤー」としてのレベル低いよ。

■以上，老プレイヤーのたわ言

まぁそれもこれも，OSIが「プレイヤーに優しくなった」おかげで，コアゲーマーではない層が進出してきてさまざまなプレイヤーが集まって――なんとなく「ネットワークRPGはこうあるべき」という慣習法のようなもので動いていた層とは違う層が入ってきたせいだろう。むろんそれは，貧困にあえぐPCゲーム界では歓迎すべきことなのだが，なかなか痛し痒しではある。

ユーザーに迎合し，ハードな顔を脱ぎ捨てて着々と「簡単に遊べる」方向へとシフトし続けるUO。それが結局のところゲーム自体をつまらないものにして寿命を縮めているということがわからないんだろうか？　この先OSIはどうするつもりなのだろう？　次々と目先を変え，すべての職業に価値を持たせ，アイテムを増やし，より難易

図15　最近のパッチで，ハイレベルなFishermanが釣れるようになったもの。お約束の「靴」もあるが，宝の地図と「ボトルに入ったメッセージ」がgood。後者は，メッセージに書かれた場所で釣りをすると，金属の箱に入ったお宝が釣れるというもの。難破船からのSOSメッセージという設定なのだが，絶対に難破船が助からないのだ

図16　かつては使い道のなかったTinkerも，いまでは多彩なアイテムを作れるようになった。かつて違法スレスレ（？）の方法で入手したようなアイテムやほかの職業で必要とされるアイテムなども次々と作れるようになっていて，存在価値が一気に高騰。筆者のGrandmaster Tinkerも，やっとこさ便利なキャラクターになったわけだ。ありがとうOSI

図17-1　Mining（採掘）のSkillも変更されて，Skill値が高いといろんな色の鉄鉱石が掘れるようになった。合計9種類あるのだが，筆者のMinerはまだSkill値88なので掘れるのはここまで

図17-2　んで，これがValorite（薄いブルー）のIngots（鋳塊）で作られたプレートメイルセット。とはいえ，Skill99で初めて掘れるものだけにさすがにツラいのか，Gloveがない。普段着（？）にはできないけど，かなりお洒落

図19　こいつが，見ると泣きたくなる黒デーモン。友達の船にいきなり出現したところをパチリ（あんがと30）

図18　家に設置するいろんな大道具（？）も設置できるようになったCarpenter。最近のパッチで，イーゼルやらベッドやらも作れるようになったので，ちょっと楽しい

図20　こいつが，2年半もの間筆者のメインキャラクターをしている「Kazuhisa」。Pacificのどこかで見かけても石投げないでほしい。いろんなSkillをGMにして，いろんなSkillを忘れていった，いろんな記憶が詰まったキャラクター。効率よく作られたわけではないのでステータスとか滅茶苦茶だが，愛着があって手放せない

図21　一時メインダンジョンにしていたDeceit。何度も何度も殺されているにもかかわらず，ついまた行ってしまう。いまだに通っているのは秘密

図22　最初に倒したときは本当に嬉しかったWater Elemental。いまでこそひとりでも余裕でホイホイだが，ゲーム開始2週間ぐらいはコワくてしかたなかった

度を下げ……そんなことが本来の「UO」の姿から離れていくことぐらいわかっているだろうに[*19]。むろん，ゲームとしての完成度は日々高くなっている。機能も増え，バグも直り，サーバの問題も次々と解決されている。初心者には優しく，上級者にも優しく。あれだけの巨大なシステムを更新し続けるのは，かなり大変な作業に違いない。

しかし。

最初に自分たちが目指したものはなんなのか。OSIにはもう一度考えてもらいたいものだ。純粋に「RolePlay」を目指して作ったのではなかったか？　いろいろと機能を追加して，職業を細分化し，初心者に甘い顔見せることが「RolePlay」なのか？　モノを作ることが楽しかった（そしてつらかった），Elementalをヒーヒーいいながら倒していたあの頃が，真のUOの姿ではないだろうか？　ここまで書いてきた数百行のことは，厳しくツラいあのUOの時代が，ちょっと懐かしくて恋しいプレイヤーのたわ言だ。しかし，きっと同感の士がいるに違いない。

なんだかんだいったところで，いろんな意味で，ここしばらくはUOを超えるネットワークRPGは登場しないと思う[*20]。いまだに進化し続け，新たな冒険者をいざない，着々とその面構えを変えていく。本当に小さな問題から，致命的な大きな問題まで，誠意を持って対応している（ように筆者には見える）OSIやEASq[*21]のおかげで，いまだに魅力の尽きない「世界」として君臨し続けている。

仕事でありとあらゆるPCゲームをプレイしている筆者だが，常にゲームマシンではUOが起動されている。去った者も多いが，ブーたれながらも残っている者も多い。いまでも筆者にとって「違う人生」の場所であり，いまさらBritanniaの土地に別れを告げるのは考えにくい。数年前に最初に買ったこの家でMageであるKazuhisaは生活し続け，そしてUltima Online2へと引き継がれていくのだろう。たかが仮想現実，たかがゲームだが，あまりにもたくさんの思い出が詰まっている場所，Britannia。願わくば，この世界が永遠に続きますように。

＊19　「本来の姿」はてめーが決めることじゃないだろ，という当たり前の意見は受けつけない。

＊20　というわけでEverQuestがちょっと未知数。筆者個人としては，あのクラスのものを受け入れるほどのPCゲームの市場は，まだ日本にはないと思う。一部のコアゲーマーが食いついておしまい。いや，もちろん本数はかなりいくと思うけど。

＊21　日本での発売元「エレクトロニック・アーツ・スクウェア」の勝手な省略表記。お願いだから「EASq」での表記を許してください。いくらなんでも名前，長すぎます（涙）。きっと名前の後半部分の会社が許してくれないんだろうなぁ。

COLUMN　オークションっていいんだっけ？

世界中に散らばっているUOプレイヤーだが，オンラインオークション「eBay」で，多数のアカウントの売買がなされている（http://www.ebay.com/）。

カタチとしては「ソフトウェア」の売買なんだけど，正確にいえば「アカウント」の売買。1年，2年とプレイされてお金も家も船もレアアイテムも持ってて，さらに「超」強力なキャラクターがいるアカウントを，高額でやり取りするわけだ。下は10ドルから，上は6000ドルぐらいまで，金額の大小は中身のデータ（というかなんというか）で決まっている。

費やす時間，苦労などを考えれば，モノによっては安い買い物かもしれない。誰とも知れぬ外人さんを信用してお金をやり取りするのはちょっと勇気がいるかもしれないが，どなたか試してみない？

というより「アカウントの売買」は禁止されているはずなのだが，一体全体OSIとEASqはどうするつもりなんだろう……。

ズラリと並んだアカウントオークション。見に行くだけでも面白い。なんぼなんでも3000ドルはどうかと思うが……。

Direct3D Retained Mode講座
2段レンダリングvsインタポレータ

菊地 功 Kikuchi Isawo

Direct3D RMを使っていきます。RMには実にさまざまな機能が装備されています。クォータニオンによる回転自動補間機能＝インタポレータを使ってみましょう。

Dreamcastが発売になってから，おそらくこの本の発売時点では4カ月以上が経過していると思うのだが，はたして着々と巻き返しを図れているのだろうか。もちろん出だしは好調だったし，グラフィックパワーは圧倒的なんだが，やはりソフトが少ないというのはコンシューマ機にとっては致命的だ。結局サターンではまともに3Dエンジンを使いこなせるのがセガだけだったというのが敗因だろう。その点，DreamcastはWindowsCEとDirectXが載るから大丈夫，というのが一般的な見解だったのだが，そっちの対応も遅れに遅れているようだし。ま，マイクロソフトさんには早いとこその辺を完成させてもらいましょ，って，なんでマイクロソフトの応援をしなきゃいけないんだ。

とりあえずDirectXが走ってくれれば，PCのソフトがそのままとはいかないまでも，最小限の手直しでDreamcastのソフトが作れることになるんだろう。で，気になるのはユーザーに開発環境を提供してくれるかって辺りなんだが，まあこれはないんだろうな。

PowerVR2だってPCに載るし，もちろんすぐそれを凌駕するパワーを持ったビデオチップも登場して，ケタが違う容量のメモリも使い放題なわけで，PCのほうが圧倒的に自由度が高くて面白いはずだから，必要ないといえばそれまでなんだが。DreamcastのDirect3DはIMしかないしね。

さて，前回はDirectX5からX6への切り換え時期にちょうどぶつかってしまったので，付録CD-ROMにはDirectX6 SDKを収録したものの，記事はDirectX5ベースとなってしまった。とはいっても，インタフェイスが新しくなったり，それにともなってちょっとメソッドが増えたりしているが，RMを使う限り，X5でもX6でもそうは変わらない。以下に新しくなったインタフェイスの主要なところを示す。

[DirectX5]	[DirectX6]
IDirect3D2	IDirect3D3
IDirect3DDevice2	Direct3DDevice3
IDirect3DRM2	IDirect3DRM3
IDirect3DRMDevice2	IDirect3DRMDevice3
IDirect3DRMFrame2	IDirect3DRMFrame3
IDirect3DRMViewport	IDirect3DRMViewport2
IDirect3DRMMeshBuilder2	IDirect3DRMMeshBuilder3
IDirect3DRMTexture2	IDirect3DRMTexture3
IDirect3DRMMaterial	IDirect3DRMMaterial2

見てわかるように，早い話が後ろについてたサフィックスがごっそりひとつ上がったと思えばよい。前回のサンプルもこの辺のインタフェイス名を変更すれば，「DirectX6対応」となる。メソッドのいくつかは引数が増えたりしているが，その辺はコンパイラに怒られたら自分でヘルプを参照してもらいたい。

ただし，IDirect3D3とIDirect3DDevice3は使わずに，IDirect3D2とIDirect3DDevice2は従来のままでよい。これは，IDirect3DRM3インタフェイスの作成時に，新しいIDirect3D3インタフェイスは指定できないからだ。

この2つはIMのインタフェイスだが，これはつまりDirectX6で新たに追加対応されたハードウェアアクセラレーションにはRMはまったく対応していないことを示す。がっっっっっ……かりである。「RMを使う限りX5でもX6でもそうは変わらない」といった意味がわかってもらえただろう。これではX6を使う意味がほとんどないし，たとえば誰かにアプリケーションを配布する場合に，X5しか入っていない（Windows 98しか入っていない）マシンでも動くということで，X5で作ったほうが親切ともいえる。

が，ここではやっぱりDirectX6を使うことにする。だって，付録CD-ROMに入ってるんだも～ん（編注：DirectX 6.1を収録）。念のためにいっとくが，古いインタフェイスもそのまま残されているので，もちろんDirectX5以前のインタフェイスで書かれたプログラムは，DirectX6 SDKでもそのままコンパイルできる。

もうひとついっとくと，IDirect3DRMTexture2インタフェイスでBMPのテクスチャを貼る場合，上下が逆さまになっていたが，IDirect3DRMTexture3インタフェイスではこれが修正された。つまり，インタフェイスを新しいものに載せ換える場合，BMPのテクスチャを使っている場合にはテクスチャの上下を入れ換える必要がある。SDKのサンプルの中ではTiger.xがBMPのテクスチャを使っているが，これを前号に収録したMeshViewで表示するとテクスチャが上下逆さまになるのはこのためだ。

今号に収録したMeshViewはDirectX6で書き直しているので，Tiger.xは正常に表示されるが，反対にDirectX5以前を想定してBMPテクスチャを使用したオブジェクトはやっぱり逆さまになる。難儀なやつだ。いっそのこと，BMPの構造自体を直してみてはどうだ？　きっと大混乱になるぞ（そういうフラグがあったような気もするが）。

ウィンドウモードとフルスクリーンモード

え～……，前回でウィンドウモードを含めたライブラリの作成と，2段レンダリングとかアニメーションを予告していたらしい。ごめんなさい。ライブラリできてません。だってだって，復刊号の原稿執筆の後遺症で2カ月くらい放心状態だったりとか，修行の旅に出てたりとか，HDDがクラッシュしたりとか，やる気が出なかった（これがいちばんか？）とかで，ほぼなにもできなかったのだ。お詫びといっちゃあなんだが，前回の立方体くるくるのサンプル（図1　D3DSmpl）をウィンドウモード対応に修正してみた（もちろんDirectX6で）。D3DSmpl.cppの最初のほう，FULLSCREENっていう定数がFALSEのときはウィンドウモード，TRUEのときはフルスクリーンになる。ウィンドウモードとフルスクリーンモードとの違いとしては，

・協調レベル
・プライマリドライバしか使えない
・フリップが使えない
・256色モード時のパレットの対応
・ウィンドウの移動/サイズ変更時の処理
・ウィンドウのクリップ

といったところかな。ウィンドウモードのほうがやることが多いのだ。

協調レベルってのは，フルスクリーンモードが「好きにやらせてもらうぞ～，うお～！」っていうのに対して，ウィンドウモードは「ごめんあそばせ。ちょっと仲間に入れてもらうわよん」って感じのアレ。

プライマリドライバってのは要するにWindowsが動いてるドライバだから，切り換えるわけにいかない（マルチディスプレイモードはとりあえず考

図1 D3DSmpl

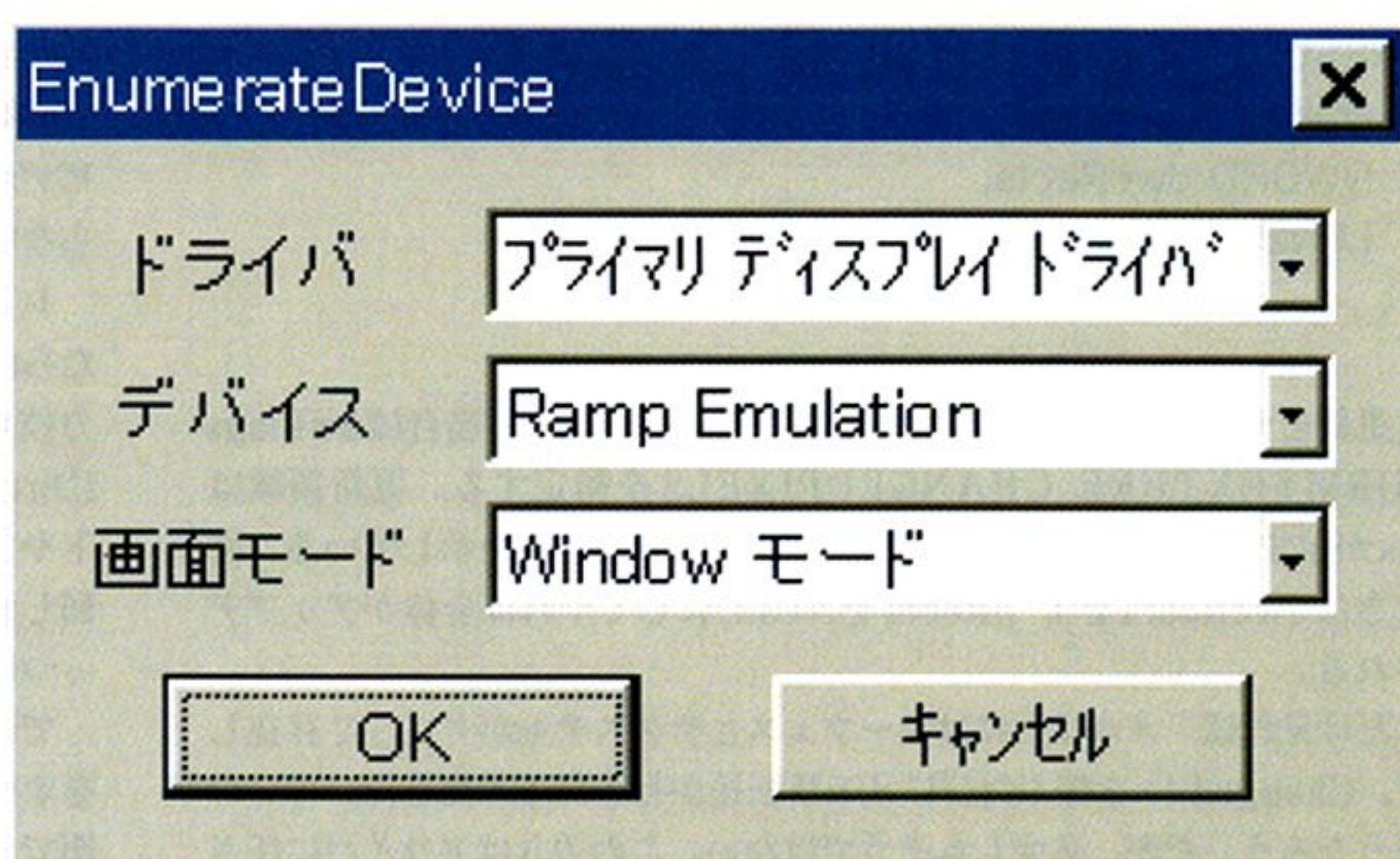



図2 EnumDevice

慮しない)。ドライバの列挙が必要なくなるので，ここだけはちょっと楽ができる。

フリップが使えないとどうするかっていうと，自分でバックバッファを作って，プライマリに転送しなきゃいけない。ウィンドウの移動時はこの転送する先の矩形を更新しなくちゃいけないし，サイズが変更されたときはサーフェスからなにからチャラになってしまうので作り直さなきゃいけない。

ウィンドウのクリップってのは，ほかのウィンドウがオーバーラップしたときに，そっちに間違って書き込んだりしないようにするもの。このうち，今回のサンプルではウィンドウサイズは固定にしているので，そのときの処理はつけていない。

あと，パレットは起動した最初のときは最適化するようになっているけど，ほかでパレットを操作されたときのことは考慮していない。真面目にやろうとするといろいろ面倒なのだ。

それと，クライアントのサイズは320×240にした。っていうのは，ウィンドウモードだと，ビデオメモリがほとんど空いていないことが想定されるから。たとえば，ビデオメモリが4MBだったとしよう。で，画面モードが1600×1200の16ビットカラー。すると，それだけで1600×1200×16ビット＝約3.7MBになり，残りは300KB強。3Dアクセラレーション(とテクスチャ)に対応したカードなら，バックバッファ(プライマリと同じビット深度=16ビット)とZバッファ(16ビット)もビデオメモリから取らないといけないから，640×480だと約1.2MB，ぜんぜん入りきらない。これが320×240ならば300KBとなり，すれすれビデオメモリに入るわけだ。この辺のサイズも定数SCREENWIDTHとSCREENHEIGHTで設定できるので，自分の環境にあわせて変更してもいいだろう。なお，ビデオカードが3Dに対応していなくて，HALが使えないのであれば，このバッファはシステムメモリに取られるので，こういった制限はない。

このサンプルでは途中で切り換えたりはできないけど，その辺はソースをちゃんと読み下してくれればできるようになる。これをスケルトンとして使ってもらえれば，ライブラリの代わりになるんじゃない？　ね，ね。

あとあと，ドライバとかデバイスとか画面モードをユーザーに選択させるダイアログのサンプルも作ってみた(図2　EnumDevice)。要はコールバック関数をぐるぐる回して，取得できるGUIDやモードを片っ端からリストボックスに放り込んでいる。この辺のことは(MFCを使わない)コントロールの知識を必要とするが，DirectXとは直接関係ないので深くは説明しない。注意が必要なのは，ドライバを変更すると，選択できるデバイスと画面モードも変わるので，その辺のリストボックスの内容も更新する必要があることだ。ダイアログを表示させるには，

```
BOOL EnumDevice ( HINSTANCE hInstance, HWND hWnd ) ;
```

をコールすればいい。OKされればTRUE，CANCELされればFALSEが戻り値として返される。肝心のデバイスなどについては，グローバルに宣言されたCDisplayParamクラスのインスタンスDisplayParamに入れられる。何度もいうようだが(いや，こっちの記事では初めてか)，クラスからは逃れられない運命にあるのだ。あきらめて理解しよう。

たとえばドライバのGUIDがほしければ，DisplayParam.GetDriverGUID()でGUIDのポインタが取得できる。だから，DirectDrawオブジェクトを作成したければ，

```
DirectDrawCreate ( DisplayParam.GetDriverGUID () ,
                                        &lpDD, NULL ) ;
```

とするだけでいい。うむ，簡単だ。

CDisplayParamクラスのメソッドについては，DeviceEnum.hをざっと眺めればだいたいのことはわかると思う。なお，ダイアログを表示する前にDisplayParamにGUIDなどを設定しておけば，表示されたときにそれに対応するドライバなどがデフォルトで選択されるようになっている。

また，ちょちょっと手を加えれば，必要な機能のないデバイスや不要な画面モードをふるい落とすことも簡単だ。このサンプルでは，テクスチャに対応していないデバイスと，8ビットカラー以下の画面モードは切り捨てるようになっている。

こいつはウィンドウモードも選択できるように作ってあるので，さっきのサンプルとうまく組みあわせて使えば，画面モードなんかもうへっちゃら，パーペキだ。ほかのアプリケーションに組み込む場合には，EnumDevice.cpp,EnumDevice.h,EnumDevice.rc,resource.hをプロジェクトに加えて，EnumDevice.cpp内のWndProc()とWinMain()関数をごっそり削ればいい。

ぴっかぴかの映り込み

前号では，周囲が映り込んでいるような質感となるクロムラップを試した。それなりにテカテカしているようには見えるのだが，このままでは周囲のオブジェクトが映り込むことはない。

この手法で周囲のものまで映り込ませようと思ったら，2段レンダリングがもっとも現実的な方法だろう。理屈としては，まず映り込ませたい風景をレンダリングし，それをテクスチャとしてクロムラップする。RMでは，サーフェスからテクスチャを作る手法として次のメソッドが用意されている。

```
HRESULT IDirect3DRM3::CreateTextureFromSurface (
    LPDIRECTDRAWSURFACE lpDDS,
    LPDIRECT3DRMTEXTURE3 * lplpD3DRMTexture3
) ;
```

なんともわかりやすい。テクスチャを作ったあとにサーフェスが更新された場合は，

```
HRESULT IDirect3DRMTexture3::Changed (
    DWORD dwFlags,
    DWORD dwcRects,
    LPRECT pRects
);
```

により更新をテクスチャに伝えてやる。ピクセルを更新する場合はdwFlagsにD3DRMTEXTURE_CHANGEDPIXELSを指定する。更新領域はpRectsが示す矩形の配列で，dwcRectsはその矩形の数を示している。面倒なときはdwcRectsを0，pRectsをNULLにしてやれば全体がアップデートされる。

これだけ見れば，メモリ上にはサーフェスとテクスチャがダブって存在していて，Changed ()を呼ぶたびにメモリ転送が行われるのだとたいていの人が思うだろう。だが，必ずしもそうではない。この辺りはドライバに任されているようなのだが，指定されたサーフェスがそのままテクスチャとして利用できるならば，コピーはされないこともある。というのは，RMではサーフェスとテクスチャはまったく違うオブジェクトに見えるが，IMでは同じもの，というかテクスチャとは「これはテクスチャで使うよ」という目印をつけられたサーフェスなのだ。

したがって，その上に乗っかっているRMでも同じことがいえる。しかも，ビデオチップやドライバによっては，その目印をつけられて作成されていないサーフェスでもテクスチャにできてしまうようだ。たとえば筆者の手持ちのMillennium G200はその辺が非常に柔軟で，テクスチャ用に作られたサーフェスであろうがなかろうが，(ビデオメモリ内に取られたサーフェスならば)テクスチャにもできるし直接レンダリングもできてしまう(かなり稀な例のようだ)。こういった場合には，CreateTextureFromSurface ()で作られたテクスチャはサーフェスと同一のものであり，したがって実はChanged ()を呼ばなくてもテクスチャは更新されている。もちろんサーフェスを作るときに「これはテクスチャだよ」といって作ってやればコピーは作られない(はず)なのだが，それだとビデオチップやドライバによってはそこへ正常にレンダリングできない場合もある。Voodooシリーズはビデオメモリとテクスチャメモリが別なので容易に想像がつくが，それ以外でもたいていの場合は失敗するらしい。この辺を柔軟に対応できるのが確認できたのは，前述のG200とVeriteだけ，Riva系やPermediaでは画面が壊れてしまった。

ということで，テクスチャとなる画像をレンダリングするのは，オフスクリーンとして作られたサーフェスであり，CreateTextureFromSurface ()でコピーが作られてしまったらそれはそれで無駄だがしかたがないということで納得するしかない。ちなみにVoodooではチップの仕様なのか，なぜかビデオメモリからオフスクリーンプレーンを取ることができない。よくわからないが，頭にきたのでVoodooは切ることにした。

レンダリングしたいサーフェスが2つある場合，どうするか。ひょっとしたらほかになにか方法があるのかもしれないが，いちばん手っ取り早いのは力技だ。つまりオブジェクトを2つずつ作る。サーフェスを指定するのはIDirect3DDevice2オブジェクトを作るときだから，それ以降のビューポートやフレームなどを2つずつ作ることになる。ただし，メッシュなどは使い回しが可能なのでひとつでOKだ(ひとつのメッシュを複数のフレームにくっつけられる)。

では世界の構築をしよう。ティーポットが世界の中心にあり，その周りをオウム貝とボールが回っていることにしよう。なんとも脈絡のない意味不明でシュールな世界だ。ティーポットはてかてかで，オウム貝とボールがそこに映り込む。

この場合，テクスチャをレンダリングするカメラはティーポットの中心に据えられることになる(図3　世界の構築)。ただし，ティーポット自身を描画する必要がないので，テクスチャ側の世界にはティーポットはいない。そうしてレンダリングされたテクスチャは，リアルスペース(？)のティーポットにクロムラップされる。テクスチャスペースとリアルスペースのオウム貝とボールは同じように動いているので，あたかもリアルスペースの周囲がティーポットに映り込んでいるように見える(図4　reflect)。このサンプル(サンプル・Reflect)は，もちろんさっきのEnumDeviceを組み込んであるので，自分で自由なドライバや解像度を選択できるが，結構ビデオメモリを食うので(HALでは)，あまり高い解像度では動かないかもしれない。テクスチャのサイズは2の累乗で，正方形が効率がいいというので，テクスチャサーフェスは256×256固定にしてある。

とまあここまでやってみたのだが，やっぱりRiva系では正常に動かないことが発覚した(その他は不明)。テクスチャのレンダリングができてないらしい。どうもレンダリングできるのは，プライマリとバックサーフェスに限られるようだ。とはいえ，HEL (Reference Rasterizer含む)ではレンダリングできるわけで，悪いのは筆者ではなくてハードウェアだ，といいきってしまおう。一度バックバッファに描いておいてテクスチャサーフェスにBlt ()で転送するというのもやってみたが，レンダリングとBlt ()が同期しないのでテクスチャが壊れてしまう(レンダリングが済む前に転送してしまう？)。ま，なんにせよG200でなんの問題もなく動くもんだから，それ以上追究する気にならんかった。自分のマシンで「うまく動かんぞー！」って場合は，自分で方法を考えて対処するか，Rampドライバに甘んじるか，G200に買い換えてくれ。ふっ，TNTやBansheeに勝ってしまった。

さて，ここでよく観察すると，不自然な点がいくつかわかるだろう。まず，映り込みの範囲が狭い。カメラは180度未満の領域しか見れないのだ

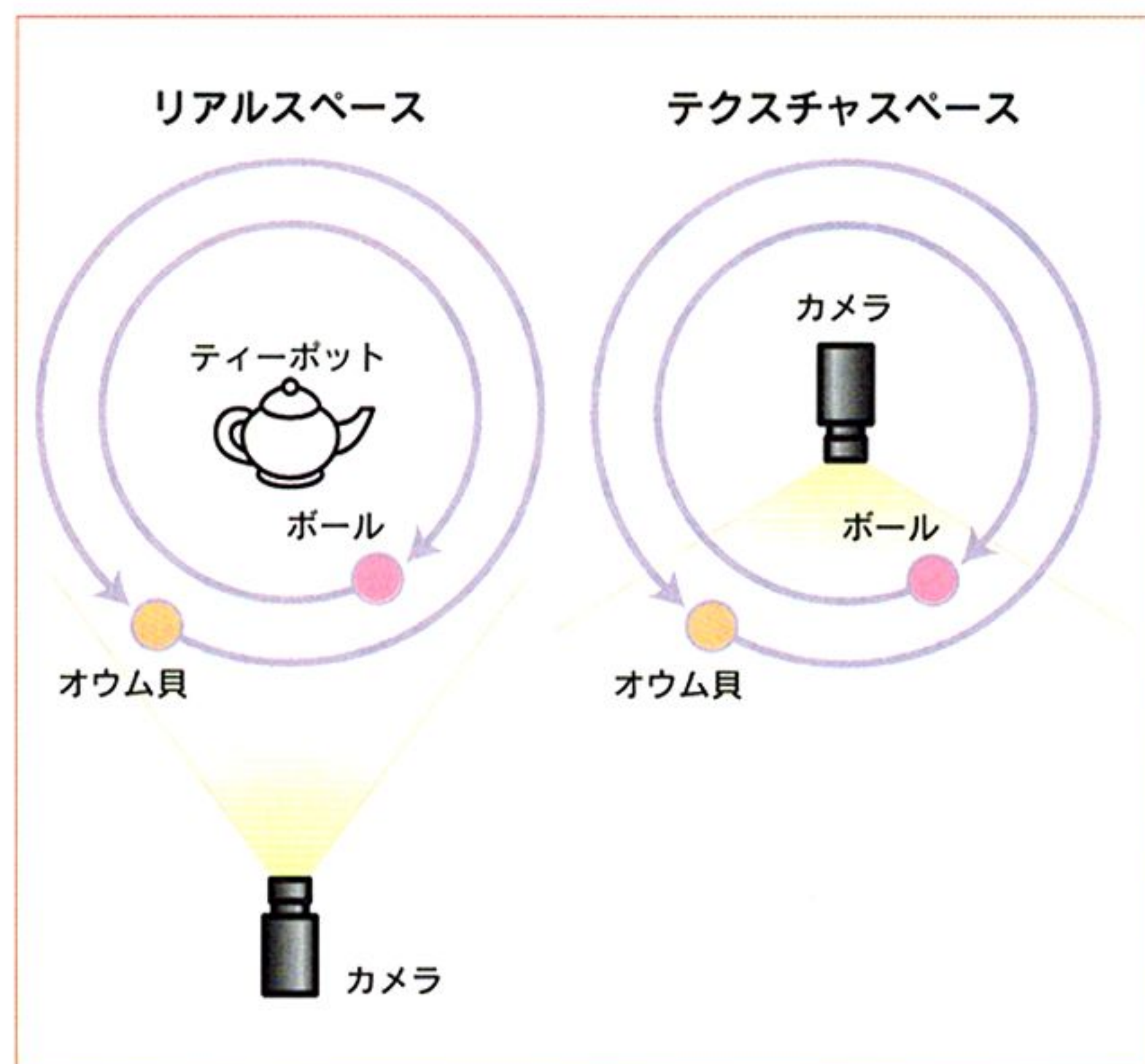



図3　世界の構築

図4　reflect

が，論理的には360度近くの範囲でテクスチャが作られなければウソになる。サンプルでは多少画角を広くしてはいるのだが，十分とはいえない。

もうひとつは，映り込んでいる物体が中心にあるときよりも，端のほうにあるときのほうが大きく映っているように見える。クロムマップで貼りつけるテクスチャが無限遠の風景であることを想定しているからという点がひとつと，実際にテクスチャは物体が中心よりも端にあるときのほうが大きくレンダリングされてしまっているからだ。これはフロントクリップによるものだ。パースペクティブ法によるレンダリングでは，物体は近いほど大きく描かれることになる。が，厳密にいえば，カメラとの距離ではなく，フロントクリップ平面との距離による。つまり，カメラを中心として周りを回る物体は，カメラとの距離が常に一定であるにもかかわらず，フロントクリップとの距離は端のほうが近くなり，結果大きく描画されてしまうというわけだ(図5　フロントクリップと物体の距離)。

フロントクリップまでの距離を物体までの距離と比べて無視できるほど小さく設定すれば解消されるが，むやみに精度を要求される値を設定すると，今度はZバッファが破綻するという弊害がある。また，このために画角を極端に広く取ることもできない。

解決法としては，たとえばテクスチャスペースでは物体の動きを直線に展開するなどが考えられる(図6　物体の回転運動を直線運動に展開)。しかしその場合，ライトも物体の動きにあわせて移動させなければ陰影がウソになるし，ばらばらに動く物体が複数あった場合はお手上げだ。もうひとつの方法は，複数のカメラを角度をつけて置き，1枚のテクスチャを複数のカメラでレンダリングする(図7　複数のカメラでテクスチャをレンダリング・ステレオグラムの記事参照)。そうすれば，周囲と中心での極端な大きさの違いは抑えられ，広い画角を得ることもできる。が，もちろんそれだけ重くもなる。まあ，クロムマップ自体がイカサマなのだから，あまり厳密に考えずに，適当に「っぽい」ってことだけで妥協するのがいちばんではなかろうか。

そもそもクロムマップで無限遠ではない物体を映り込ませること自体に無理があるんだし。「周囲を回らせる」というのが特に不自然な点を目立たせているというのもあるのだが。

アニメーション

3Dでアニメーションというと，フレームアニメーションを指すのが一般的だ。要するにオブジェクト自体は変化せずに，フレームだけを移動させたり回転させたりする。それに対してモーフィングみたいなものもアニメーションの範疇だが，Direct3D RMではこちらはインタポレータというもので処理できる。いってみれば，Zガンダムの変形は(フレーム)アニメーションだが，ゲッターロボの合体はインタポレータである。まずは簡単なフレームアニメーションから行ってみよう(以下単にアニメーションといった場合はフレームアニメーションを指す)。

アニメーションにはIDirect3DRMAnimation2とIDirect3DRMAnimationSet2というインタフェイスを使う。前者はフレームの動作を直接指定するインタフェイスであり，後者は複数のIDirect3DRMAnimation2オブジェクトを束ねるインタフェイスである。IDirect3DRMAnimation2自体はひとつのフレームに対するアニメーションしか設定できないが，IDirect3DRMAnimationSet2に複数のIDirect3DRMAnimation2オブジェクトを接続してやれば，複数のフレームのアニメーションを一度に制御できるというわけだ。

ちなみにこのIDirect3DRMAnimationSet2はMeshViewでも使っている。Xファイルに格納されたアニメーションデータを読み込みたい場合には，IDirect3DRM3::CreateAnimationSet ()でアニメーションセットのオブジェクトを作成したあと，IDirect3DRMAnimation2::Load ()でファイルを指定すれば，アニメーションに設定されたフレームやメッシュごとごっそりメモリにロードされる。あとはIDirect3DRMAnimation2::SetTime ()で時間を指定すれば，勝手にその時間の姿勢になるというわけだ。なんとこれだけ。リニアに順送りしたければSetTime ()に渡す値を少しずつ加算していけばいいし，減算すれば逆送りにもできる。MeshViewのソースも参考にしてほしい。

アニメーションデータのファイルを作るのも，3Dグラフィックソフトを使うのが一般的だ。そういったソフトでは，インバースキネマティクスと呼ばれる機能でアニメーションを定義できる。それをファイルに落とし，必要があればXファイルにコンバートすることで，Direct3D RMで利用できるアニメーションデータを作ることができる。まあ，そういったソフトの使い方についてはそちらのマニュアルに任せるとして，ここではソフトを使わずにプログラミングだけでアニメーションさせてみよう。

とりあえずトーフロボットの腕である(図8　トーフロボットの腕)。メッシュ自体はcube.xを使い，上腕・下腕はそれをY軸方向に引っ張ってい

図5　フロントクリップと物体の距離

図6　物体の回転運動を直線運動に展開

図7　複数のカメラでテクスチャをレンダリング

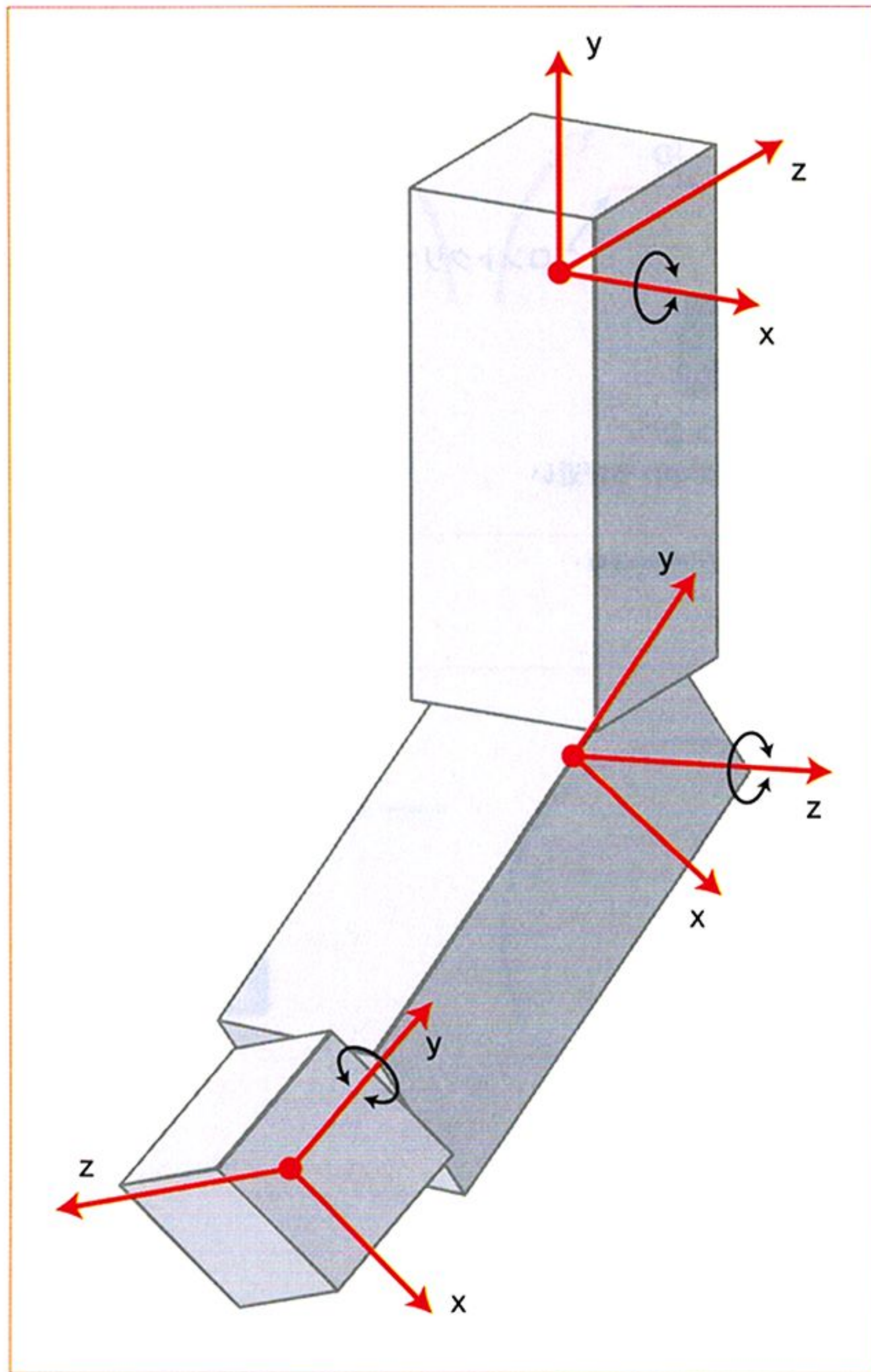



図8　トーフロボットの腕

る（正確にはX・Z軸方向に縮めている）。座標軸はそれぞれの部位のフレームであり，それぞれのフレームを次のようなタイムスケジュールで動かしてみる。

部位	時間	動作
上腕（Shoulder）	0〜10	X軸周りに-0.6ラジアン
下腕（Arm）	5〜15	X軸周りに-1.0ラジアン
手（Hand）	10〜20	Y軸周りに適当に何回転か

このとき，下腕は上腕の，手は下腕の子フレームである。そうしておけば，上腕を回転させたとき，その子フレームである下腕と，さらにその子である手のフレームは勝手についてくる。ではまず上腕のアニメーションからセットしていこう。

まずはIDirect3DRMAnimation2オブジェクトを作成し，それにフレームをセットする。

```
lpD3DRM->CreateAnimation ( &animation ) ;
animation->SetFrame ( lpD3DRMShoulder ) ;
```

animationはIDirect3DRMAnimation2のポインタであり，lpD3DRMShoulderは上腕のフレームだ。

アニメーションを設定するには，時間キーと動作を指定すればいい。時間キーというのは，いまの場合は0と10に当たり，0のときに0ラジアン，10のときに−0.6ラジアンと設定すれば，SetTime()で時間を5に指定したとき，そのキーを補間して−0.3ラジアンの回転を設定してくれる。アニメーションに回転を指定するには，IDirect3DRMAnimation2::AddRotateKey()メソッドを使う。しかし，このメソッドでは回転をD3DRMQUATERNION構造体で表されるクォータニオンという形式で指定する必要がある。

```
typedef struct _D3DRMQUATERNION {
    D3DVALUE  s;
    D3DVECTOR  v;
} D3DRMQUATERNION;
```

D3DVECTOR構造体というのはベクトルを表す3つの実数からなっているので，クォータニオンはそれにsを加えた4つの実数のセットである。いろいろと理屈があり，フレームの姿勢を表すのに，本来はX軸，Y軸，Z軸の向きをそれぞれベクトルで表す，つまり9つの実数が必要であるのに対して，クォータニオンならば4つの実数で済む，ということらしい。日本語でいうと四元数ということになるが，クォータニオンのほうがかっこいいのでそう呼ぶことにしよう。さて，このクォータニオン，筆者も勉強不足でよくわからない。「早い話がvで示されたベクトルの周りをsラジアンだけ回転させるんだろ」と思ったのだが，どうもそうではないらしい。だが心配はご無用。Direct3Dには回転をクォータニオンに変換する関数が用意されている。

```
LPD3DRMQUATERNION D3DRMQuaternionFromRotation (
    LPD3DRMQUATERNION lpquat,
    LPD3DVECTOR lpv,
    D3DVALUE theta
) ;
```

lpquatは格納されるクォータニオン構造体へのポインタ，lpvは回転の軸となるベクトル，thetaは回転角を表す。これ以外にもクォータニオンから回転へ変換する関数や，クォータニオンの合成を行う関数などが用意されているので，とりあえずD3DRMQUATERNIONの各メンバに対して直接手を下す必要はない。フレームの姿勢を制御するのに，クォータニオンはなんとなく都合がいいらしいってことさえ覚えておけばきっと大丈夫だ。

さて本題に戻ろう。X軸回りに回転させるので，軸のベクトルは{1, 0, 0}である。時間0では回転は0であるから，そのときのクオータニオンは，

```
D3DRMQUATERNION rqQuat;
D3DVECTOR vect;
vect.dvX = D3DVAL (1.0) ;
vect.dvY = D3DVAL (0.0) ;
vect.dvZ = D3DVAL (0.0) ;
D3DRMQuaternionFromRotation
                    ( &rqQuat, &vect, D3DVAL (0.0) ) ;
```

で取得できる。っていうか，さすがにこの場合はクォータニオンの各メンバに0を代入しておけばいいのだろうが，念のため。これをキー0の回転として設定する。

```
animation->AddRotateKey ( D3DVAL (0) , &rqQuat ) ;
```

同様にキー10の設定は，

```
D3DRMQuaternionFromRotation
                    ( &rqQuat, &vect, -D3DVAL (0.6) ) ;
animation->AddRotateKey ( D3DVAL (10) , &rqQuat ) ;
```

となる。軸ベクトルに値を入れないのは，前の状態が残っているからというのはいうまでもない。キーの設定ができたら，アニメーションセットに追加する前に，IDirect3DRMAnimation2::SetOptions()でオプションを設定しておこう。とりあえず引数としてD3DRMANIMATION_OPENとD3DRMANIMATION_SCALEANDROTATIONを指定すればいいだろう。前者はアニメーションが1回きりの再生を示す。1回きりというのは別に一度再生したらアニメーションデータが解放されてしまうとかいうことじ

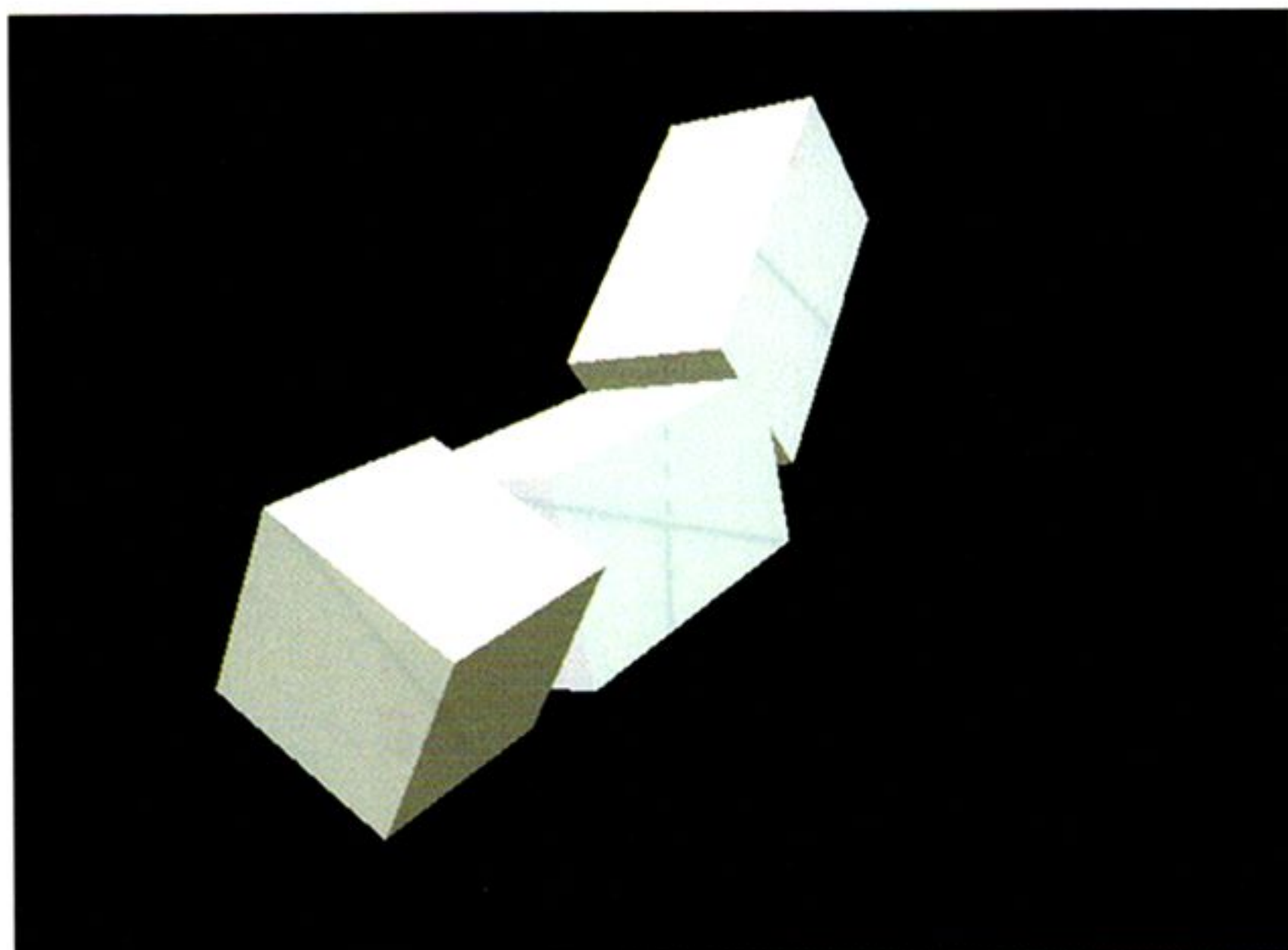



図9 サンプル・Animation

ゃなくて，時間が0以前あるいは10以降はアニメーションしないということ。もうちょっと正確にいうと，0以前が指定されたときは0のときの状態を，10以降なら10の状態を使用する。これをD3DRMANIMATION_CLOSEDにすると，繰り返し再生を行う。

いまの場合は20までは10の状態のままで止まっていてほしいので，オープンにしておく。後者はスケーリングとローテーションのアニメーションが有効ですよ，というフラグだ。D3DRMANIMATION_POSITIONを指定すると，フレームのポジションも有効になる。

さらにいえば，ポジションを設定したときは，D3DRMANIMATION_LINEARPOSITIONまたはD3DRMANIMATION_SPLINEPOSITIONで，キー間のポジションをリニアに補間するか，Bスプラインで補間するかを指定できる。いまは回転だけなので，ポジションの指定は不要だ。最後はIDirect3DRMAnimationSet2::AddAnimation ()でアニメーションをアニメーションセットに追加してやればよい。下腕や手も同様だ。ただし，隣りあうキー間でπ以上の回転を指定すると，間違えて逆回転してしまうので，何回転もさせたい場合にはキーを刻まなくてはならない。手の場合はキーを2ずつ刻んで，ついでになんとなく加減速させてみた。

あとはレンダリングさせる前に時間を設定すればいい。

```
lpD3DRMAnimationSet->SetTime ( time ) ;
time = time+D3DVAL (0.1) ;
if ( time>D3DVAL (20) ) time = D3DVAL (0) ;
lpD3DRMCamera->LookAt ( lpD3DRMArm, lpD3DRMScene,
                                   D3DRMCONSTRAIN_Z ) ;
```

0.1ずつ時間を進め，20まで行ったらまた最初から繰り返している。ここでちゃんと現実に即した経過時間を入れてやれば，ビデオカードの性能によらずに一定のスピードで再生できることはいうまでもない。最後の1行はアニメーションとは直接関係ないが，カメラを常に下腕に向けるためのオマジナイだ。このIDirect3DRMFrame3::LookAt ()というのは，フレーム(のZ軸)を，指定したフレーム(の原点)に向くように回転させるメソッドで，カメラフレームで使えば，常にそのフレームを画面の中心に捉えてくれる。lpD3DRMSceneってのは基準となるフレームということだが……なんのことやらよくわからない。最後のD3DRMCONSTRAIN_ZってのはZ軸周りには回転させませんよってこと。つまりX軸とY軸周りの回転だけで，lpD3DRMArmを画面に捉えるようになっている。これらが第2引数のフレームの軸なのかと思ったが，そうでもないらしい。ただ，まったく別のフレームを指定すると挙動不審になるのでなにかに使われてはいるようなのだが。とりあえずシーンフレームを指定するようにしておけば間違いはないようだ。

ソースは少々煩雑になっているが，ここで説明した以上の難しいことはなにひとつやっていないので，読み下すのは簡単なはずだ(図9　サンプル・Animation)。実行すると，腕らしきものがぐいっと持ち上がって，こぶしらしきものがぎゅぎゅるっと回転する。よく見ると，微妙なカメラフレームもちゃんと観察できるはずだ。それだけのものだが，フレームアニメーションってのはこれだけのものなのだ。

インタポレータる

モーフィングはインタポレータでやるといったが，インタポレータはなにもモーフィング(つまりメッシュの変形)だけしかできないわけではない。アニメーションのようにワンアンドオンリー(だがアニメーションの効果としてはもっともよく使われる)なものではなく，メッシュや背景，ライトの色を徐々に変化させたり，質感をだんだんざらざらからつるつるにしていったり，画角を変えることでズーミングしたりと用途は多岐にわたる。

が，その分扱いはちょっとだけ厄介だ。たとえばメッシュをインタポレータで操作したい場合，まずインタポレータに操作法を教えるダミーのメッシュを作り，それをインタポレータに示してやる。次にそのダミーに対して通常のメッシュ操作を行うが，実際にはその操作はメッシュではなくインタポレータに対して適用される。このメッシュはインタポレータに操作を教えるためだけに存在するので，それをフレームに貼りつけて表示することはできない。実際に再生するには，別の本物のメッシュを用意し，操作を教え込んだインタポレータと接続しなければならない。イメージとしては，工業用ロボットに動きを教えるのに，熟練工がアームを持って動かすトレース用のロボットと，実際に作業を行うロボットが別にあるようなものだ。

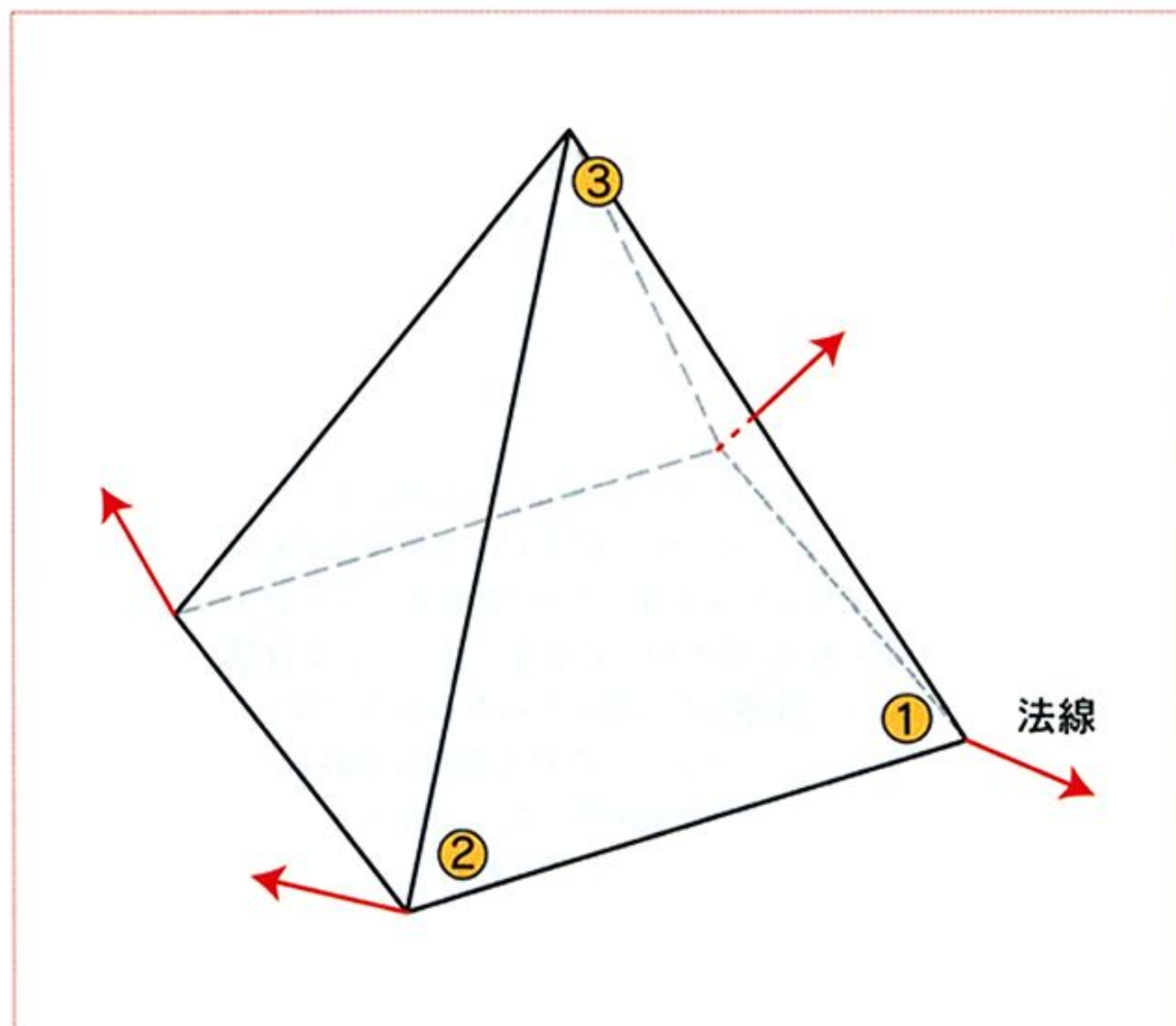
もうひとつ厄介な問題があるとすれば，それはメッシュを扱うのにIDirect3DRMMeshBuilder3ではなく，IDirect3DRMMeshを使わなければならないということだ。いままでIDirect3DRMMeshBuilder3しか使ってこなかったのはそっちのほうが楽チンだったからなのだが，IDirect3DRMMeshとて単にローレベルで使いにくいだけかというと，必ずしもそうではない。IDirect3DRMMeshならば，ひとつのメッシュ内でフェースをグループ分けし，それぞれ異なるカラーやマテリアルを設定できるのだ。IDirect3DRMMeshBuilder3::CreateMesh ()を使えばIDirect3DRMMeshBuilder3からIDirect3DRMMeshオブジェクトを作ることもできるが，操作自体はIDirect3DRMMeshメソッドしか適用されないので，いずれにせよインタポレータでメッシュを扱いたければIDirect3DRMMeshもマスターしなければならない。ちなみにインタポレータでもフレームのポジションやローテーションを再現できるので，アニメーションの機能を包含しているといえる。

というわけで，インタポレータをいじくる前にまずIDirect3DRMMeshでメッシュを作ってみよう。オブジェクトを作成するには，IDirect3DRM3::CreateMesh ()メソッドを使う。ちょっと不思議なのは，頂点数や面数といっしょに，まず頂点インデックスによって示されたフェースデータを指定するのだ。この時点で，頂点や法線を格納する領域がメッシュオブジェクト内に確保され，0で初期化される。そのあと，頂点座標や法線ベクトルをセットすることになる。普通ならまず頂点を設定してから，そのインデックスでフェースを指定させるのが自然だと思うのだが，あえてこのような手順にしたのは，まさしくいまやろうとしているような，頂点座標を後から変更できるようにするためだろう。次のような感じだ。

```
LPDIRECT3DRMMESH mesh;
lpD3DRM->CreateMesh ( &mesh ) ;
mesh->AddGroup ( 5, 4, 3, fData, &id ) ;
mesh->SetVertices ( id, 0, 5, vartices ) ;
```

AddGroup ()の引数は，先頭から頂点の数，フェースの数，フェース辺りの頂点の数，頂点インデックスで示されたフェースデータへのポインタ，グループIDを格納するポインタだ。メソッドの名前からわかるように，AddGroup ()というのは新たにグループを作成して，メッシュに追加する。上の例では5個の頂点と，4つのフェースを持つグループが追加される。それらのフェースは3つの頂点で作られる，つまりすべて三角形ポリゴンとい

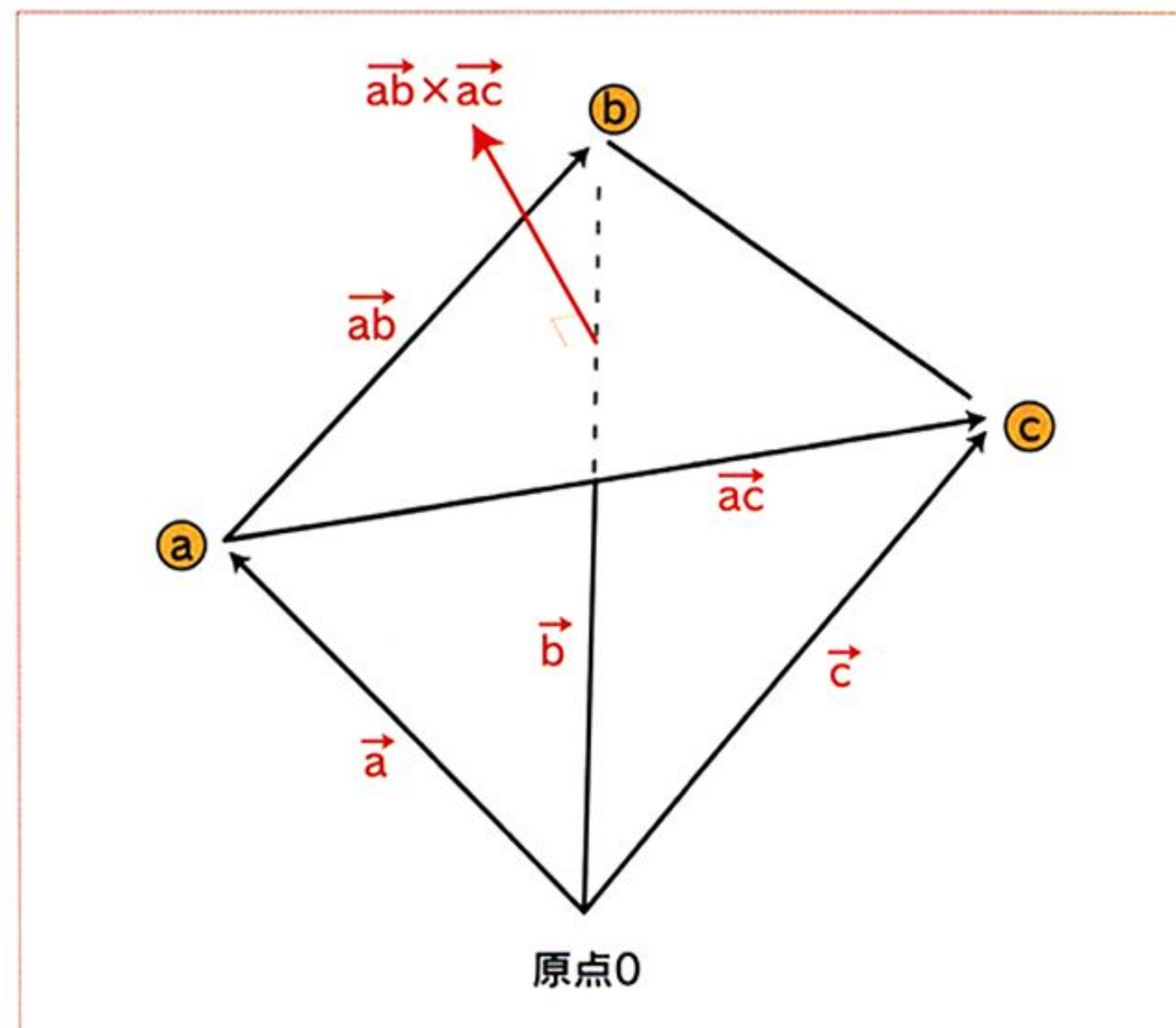
図10　四角錐のフェース

図11　法線の計算

うことだ。この第3引数で3以上の値が指定された場合は，fDataにはフェースを作る頂点インデックスをずらずらと連続して格納する。第3引数が0であった場合は，フェースの頂点数が一定ではないとみなされ，fDataにはまず最初のフェースの頂点の数，続いてその数だけの頂点インデックス，その次のフェースの頂点の数，頂点のインデックス……といったように格納する必要がある。SetVertices ()は第1引数で指定されたIDのグループの，第2引数で示した頂点インデックスから，第3引数で指定した個数だけ，頂点データをセットする。頂点データはD3DRMVERTEX構造体の配列で渡すことができる。

```
typedef struct _D3DRMVERTEX{
    D3DVECTOR position;
    D3DVECTOR normal;
    D3DVALUE  tu, tv;
    D3DCOLOR  color;
} D3DRMVERTEX;
```

positionは頂点の座標，normalは頂点の法線ベクトル，tu,tvはテクスチャ座標，colorは頂点の色である。まただた面倒なことだ。ちなみにcolorはワイヤフレームのときに使われる色で，ソリッド(ポリゴン)表示のときには使われない。テクスチャを貼らないのなら，tu,tvもとりあえずほうっておいてよい。では法線はというと，手で入力するのは大変なので，プログラムで計算することにしよう。たとえば，四角錐を作るとする。頂点数は5で，側面数は4だ(図10　四角錘のフェース)。だが，頂点を複数の面で共有してしまっては，法線も共有することになり，稜線がなまってしまう，というのは前号で説明した。そのため，頂点の座標が同じでも，異なる法線を立てたければ頂点は分けなければならず，普通に考えると頂点は12必要である。しかし，ここで少しコスい手がある。フラットシェーディングの場合，面の法線は実はフェースの最初の頂点の法線が使われる。つまり，そのフェースに関しては，それ以外の法線はどうでもいいのだ。つまり，図のように頂点に法線を立てて，フェースの指定のときにそれぞれのフェースで正しい法線を持った頂点インデックスを最初に指定してやれば，頂点は共有してもフェースの法線は破綻しない。

では，法線はどうやって計算するか。ここからは純粋な数学である。頂点abcからなる三角形があったとすると，ベクトル$\vec{ab}$は$\vec{b}-\vec{a}$で算出できる(図11　法線の計算)。同様に$\vec{ac}$は$\vec{c}-\vec{a}$である。ここで，ベクトル$\vec{ab}$と$\vec{ac}$の外積を求めれば，ともに直行するベクトルが得られる。覚えてるか？　高校で習ったと思うが。筆者は綺麗さっぱり忘れていたよ。

最後にそのベクトルの単位ベクトルを求めれば，法線ベクトルのできあがりというわけだ。さーてさて，押し入れの奥のダンボール箱にしまった高校数学の教科書を掘り出さなきゃ，って心配はない。この辺の計算をしてくれる関数をRMは用意してくれているのだ。

・ベクトルの減算

```
LPD3DVECTOR D3DRMVectorSubtract (
    LPD3DVECTOR lpd,
    LPD3DVECTOR lps1,
    LPD3DVECTOR lps2
);
```

・ベクトルの外積

```
LPD3DVECTOR D3DRMVectorCrossProduct (
    LPD3DVECTOR lpd,
    LPD3DVECTOR lps1,
    LPD3DVECTOR lps2
);
```

・単位ベクトル化

```
LPD3DVECTOR D3DRMVectorNormalize (
    LPD3DVECTOR lpv
);
```

ま，実際たいした演算ではないが，この3つの関数を組み合わせれば，フェースの単位ベクトルを算出できるだろう。できたメッシュは，いつもどおりフレームにくっつけてやれば表示完了だ。

さて，当初の目的をちょっと忘れてしまいそうだったが，そうそう，インタポレータろうと思ってたんだっけ。メッシュを制御するには，まずメッシュインタポレータを作る。そのあと，そのインタポレータからQueryInterfaceしてIDirect3DRMMeshインタフェイスを取得する。

```
lpD3DRM->CreateObject ( CLSID_CDirect3DRMMeshInterpolator, 0,
        IID_IDirect3DRMInterpolator, (void **) &lpD3DRMInterp );
lpD3DRMInterp->QueryInterface
                ( IID_IDirect3DRMMesh, (void **) &mesh );
```

まあ見たまんまなので，説明はいらんだろう。ここで取得したメッシュのインタフェイスは，インタポレータに操作をトレースさせる例のダミーである。そのあと，インタポレータにキーを設定し，ダミーメッシュを操作する。

```
lpD3DRMInterp->SetIndex ( D3DVAL (0) ) ;
mesh->SetVertices ( 0, 0, 5, vertices ) ;
mesh->SetGroupColorRGB ( 0, D3DVAL (1.0) ,
                          D3DVAL (0) , D3DVAL (0) ) ;
```

この操作が指定したキーチェインに加えられる。ここで，メッシュにグループを作らないで頂点を設定していいんかいな，と思うかもしれない。いいのだ。なぜなら操作を覚えさせることだけを目的としたダミーだから。キーを進めながら適当に頂点や色を設定したら，今度はちゃんとしたメッシュを作って，インタポレータに接続する。ダミーのメッシュはリリースしてしまって構わない。

```
lpD3DRM->CreateMesh ( &mesh ) ;
mesh->AddGroup ( 5, 4, 3, fData, NULL ) ;
mesh->SetGroupQuality ( 0, D3DRMRENDER_FLAT ) ;
lpD3DRMInterp->AttachObject ( mesh ) ;
```

こっちは本物のメッシュなので，あらかじめAddGroupしておかなければならない。グループIDは0から始まるインデックスであることがわかっているので，律儀に取得する必要はない。ダミーのほうは取得できなかったんだしね。あとは適当にレンダリングサイクルで，

```
lpD3DRMInterp->Interpolate ( time, NULL, D3DRMINTERPOLATION_
                 CLOSED|D3DRMINTERPOLATION_SPLINE ) ;
```

とでもすればいい。

ちなみにAttachObjectせずに，Interpolate ()の2番目の引数でmeshを指定してもいい。これだけ。できちゃえばどうってことないだろう。むしろメッシュのほうが手間だったね(図12　サンプル・Interpolator)。これまた意味不明な物体が色を変えながらうねうねするが，どうも色の補間はかなりお馬鹿みたいで，結構ちらつく。色のアニメーションは自分でやったほうが綺麗にできるだろう。

今回はメッシュしかやらなかったけど，フレームやビューポートのインターポレータも基本的に使い方は同じだ。ただし，すべてのメソッドがインターポレータに対応しているわけではない。たとえば，IDirect3DRMFrame2::SetSceneBackgound ()で背景の色をアニメーションさせることはできるが，IDirect3DRMFrame2::SetSceneBackgoundImage ()で背景をオーバーラップさせることはできない。あと，想像はつくだろうが，あまりでかいメッシュをモーフィングさせようと思ったら，激しく重くなるだろう。なんたって，頂点全部を補間演算することになるから。この辺はCPUがやるわけだから，いくらビデオカードが速くてもあまり変わらない。

なんて素敵なヘルプ

DirectX6になって，SDKのヘルプがCompiled HTML (だっけ？)になった。これがなんとも素敵。すさまじくボリュームのあるページは，表示し終わるまでたっぷり10秒以上は頭を休める暇を与えてくれるし，メソッドのリンクをクリックすると，そのメソッドのあるページの先頭に飛んで，スクロールして探しているうちに「あー，こんなメソッドもあったんだ」と気づかせてくれる。しかも検索も効かなかったりして，「早くヘルプから自立しようね」と促してくれているようだ。

おかげでプログラミングの腰を折られて，時間を忘れることもなく，食事を抜いたり睡眠を削って体を壊す心配もない。なんとも親切だなーおい。3DのアクセラレータよりもHTMLのアクセラレータ作ったほうがいいんじゃないか？　まあ，DirectX7にはDirectHTMLが入るというし(うそ)。日本語に訳されたマニュアルも買ってみたけど，ビデオカードが楽に買えてしまうような値段のうえに，電話帳並みの破壊力(電話帳はタダなのに)。きわめつけはハナモゲラな翻訳。そのままではどうしようもないので5冊に分離してみたが……なんとかしてくれ〜。

Interpolator
Device...(D)

Interpolator
Device...(D)

Interpolator
Device...(D)

Interpolator
Device...(D)

Interpolator
Device...(D)

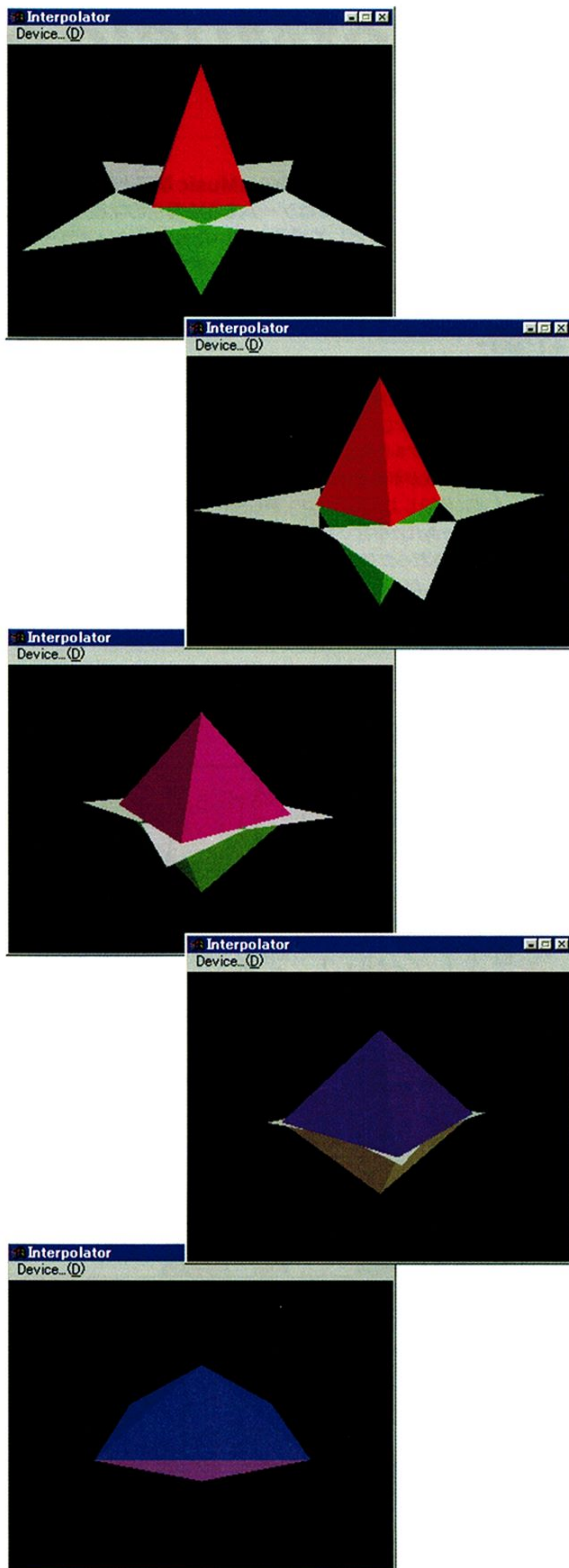



図12　Interpolatorの動作

DirectMusicを使う

永島ひとし　Nagashima Hitoshi

読者投稿によるDirectMusic解説と制御サンプルです。DirectX6.1から加えられた音楽再生用のAPI，DirectMusicはゲームBGM再生の新機軸として打ち出されたものです。従来，MIDIプレイヤーやMIDIマッパーとして機能していたものがサンプラーとして定義され直した感じでしょうか。シーケンス制御は当然としてダウンローダブルサウンドにも対応しています。

はじめに

皆さんはDirectX6 SDK（本誌復刊記念号の付録CD-ROMにも収録されていましたが）に含まれているDirectMusicのサンプルを実際に試してみましたか？　私はサンプルを試しているうちに感動で胸がいっぱいになってしまい，不本意にも涙がボロボロこぼれてしまいました。

そういえば以前同じようにパソコンから流れてくる音楽に感動したことがあったっけ……あれは中古で手に入れたX68000で初めてドラゴンナイト4をプレイしたときだったなぁなどと，がらにもなく昔のことを懐かしく思い出してしまいました。

さて，DirectX最新のコンポーネント群であるDirectMusicですが，ゲームなどに実装してみた場合，これまでのMIDIによる実装と比べてどんなメリットがあるのでしょうか？　SDK内のサンプルコードを参照しながら，投稿作品での状況についてレポートしてみたいと思います。

作者よりの注意

これは投稿レポートです。投稿者はアマチュアであり事実と異なる間違った内容を記述している可能性があります。実際のコーディングにおいては，自らレポートの内容を再度検証してみることをおすすめします。

投稿作品，２つの「mari-p」

今回の投稿作品は2つの異なるバージョンの「mari-p」というゲームです。これは私が初めて作ったゲーム作品ですが，本誌投稿用に多少手を加えてあります。ゲームの内容自体はあまりにもショボイので，DirectMusicと

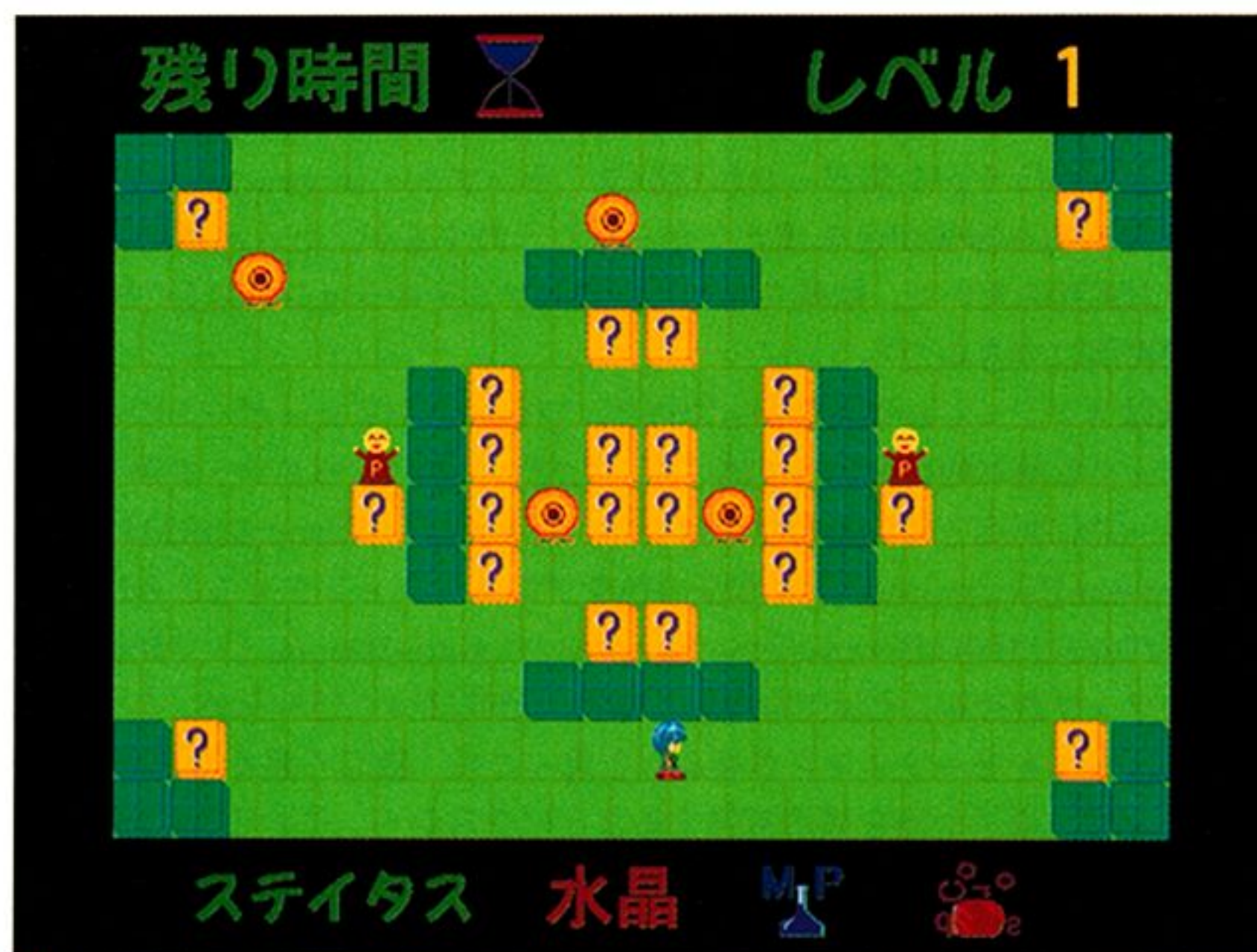



図1　サンプルゲームmari-p

図2　MIDI stream版とDirectMusic板があります

MIDI Streamの比較サンプル用と思っていただければ幸いです。
「mari-p」のDirectMusic版は「Music」フォルダ内に，MIDI Stream版は「Midi」フォルダ内にそれぞれ収録してあります。フォルダ内の「Readme.txt」を読んで動作環境等確認の上，起動してみてください。また，別の比較用サンプルとして，音源「Roland SC-55mkII」が直接生成する音を録音して，その一部をwaveデータ（ファイル名：MpSC55ad.wav）として収録してあります。こちらもぜひ試してみてください。

DirectMusic登場以前（mari11.exe）

DirectMusicが登場するまでのWindowsゲームにおける効果音・BGMの実装方法としては，効果音：「wave」，BGM：「MIDI」という組み合わせが一般的でした。実際に「mari11.exe」でも効果音出力には「DirectSound」，BGM再生には「MIDI Stream」を採用しています。

ただ，MIDIによるBGM再生にはゲームを実行する環境によって大きな差が生じるという重い足かせがあります。

加えて，MIDIの再生にあたっては，システムエクスクルーシブメッセージの送信というやっかいな問題もあり，さらに，まずいことに「DirectSound」と「MIDI Stream」のデフォルトの出力デバイスはコンフリクト（衝突）してしまうという問題（注1参照）もコーディング時に発生しました。

これはWindows内のマルチメディアのプロパティなどで事前に調整しておくことで回避できるらしいのですが，ハードウェアに詳しくないユーザーにそのような話をしても「いったいなんのことやら？」という感じで個別に正しく対処してもらわなければならないのには疑問が残りました。

そのため「mari10/mari11.exe」では「MIDI Stream」出力デバイスが「DirectSound」とコンフリクトする場合には，出力先を自動的に別の「MIDI」出力デバイスに切り替える……という方法を取っています。

Direct Musicには，このような手間のかかる問題をまったく気にせずにゲームにBGMが実装できる！　という大きな魅力があります。

注1
DirectSoundを使うための「DirectSoundCreateおよびSetCooperativeLevel」とMIDI Streamを使うための「midiStreamOpen」をプログラム内で順番に呼び出した結果は次のようになりました。なお，DirectSoundCreateの最初の引数（サウンドデバイスを表す）はNULL（デフォルト）で，midiStreamOpenの2番目の引数（同じくOpenするためのデバイスを表す）はMIDIマッパーです。

①先：DirectSoundCreate（レベル：DSSCL_NORMAL）
　後：midiStreamOpen
　　→DirectSound・・○，
　　　midiStreamOpen・・×
②先：midiStreamOpen,
　後：DirectSoundCreate（レベル：DSSCL_NORMAL）
　　→midiStreamOpen・・○
　　　DirectSound・・×
③先：midiStreamOpen
　後：DirectSoundCreate（レベル：DSSCL_EXCLUSIVE）
　　→midiStreamOpen・・○，
　　　DirectSound・・△（エミュレーションでOpenか？　再生にズレが生じるためゲームには不適）

私がこの問題に取り組んでいたのは4月（1998年）のことなので，結果の正当性について確信は持てませんが，「mari-p」ではこの状況に対処するための解決方法として，まずDirectSoundを生成，次にMIDI StreamをデフォルトのMIDIマッパーでOpen，失敗したら出力可能なデバイスをチェックして再度Openという処理を行っています。

もしかしたらこの状況を解決するためのうまい方法があるのかもしれませんが，私は見つけ出すことができませんでした。

また，なにもMIDIStreamに固執しなければ（たとえば高水準のAPIを使うとかすれば）問題はなかったのかもしれません。

いずれにしても，DirectMusicを使う限り，このような問題が発生することはなくなりました。MIDIStreamを使ったプログラミングに興味のある方には次の資料・書籍が役に立ちます。

―資料―
DirectX5 SDKのMIDI STREAM PLAYERサンプルコード
―書籍―
「Windows95ゲームプログラミングDirectX入門」：マイケル・モリソン＋ランディ・ウィームズ著，プレンティスホール出版
「Windows95 APIバイブル3 ODBC，マルチメディア編」：Richard J.Simon，Tony Davis，John Eaton，R.Murray Goerlz著，翔泳社

図3　DirectMusic Producerの起動画面

DirectMusicを使うための準備（DirectMusic Producer）

ゲーム/アプリケーションなどでDirectMusicによるBGMの実装を行うには，まず準備として手持ちのMIDIデータをDirectMusic Producer（図3）で編集しておくことが必要になります。拡張子「mid」のついたスタンダードMIDIファイルからDirectMusicで再生可能な拡張子「sgt」のついたセグメントファイルを作成するにはDirectMusic Producerを使用します。

DirectMusic ProducerをまだインストールしていないDirectX6-Extras-Dmprducerフォルダ内のSetup.exeを実行してインストールを完了させてください。

midからsgtへ変換する手順はだいたい次のようになります（手順は同じでなくても構いません），初めのうちは感触がつかめないかもしれませんが慣れてきたら各自でいろいろ試してみてください。

① まず，プロジェクトを作成します，File-Newを選択すると図4のようなダイアログボックスが現れるのでProjectを選択します。
② 同じようにFile-NewでプロジェクトにBandを追加します（図5）。
③ プロジェクトを右クリックすると，図6ようなメニューが現れるのでここでMIDIファイルのインポートを選択して，手持ちのMIDIファイルをプロジェクトに追加します。
④ インポートされたMIDIデータをクリックすると，セグメントとして次のような内容になっているのがわかります（図7）。インポートしたMarip8はパートごとの楽譜データのみにしてあるため，bandトラックは生成されません。そのためEdit-Add New Track(s)で図8のようにbandトラックを追加します（チャンネルごとの楽器などの設定は再調整が必要となるのでインポートするMIDIファイルからは削除しておいたほうがいいと思い，私の場合は削除したものをあらかじめ作っておきました）。
⑤ bandでチャンネルごとの楽器，ボリューム，パンなどの調整をします。band編集用画面（図9）の座標は，x：パン，y：ボリュームとなっています。調整した結果は「Copy/Paste」でセグメントのbandトラックに貼り付けることができるので（図10）bandデータを貼り付けたら，とりあえず曲を一度聞いてみます。

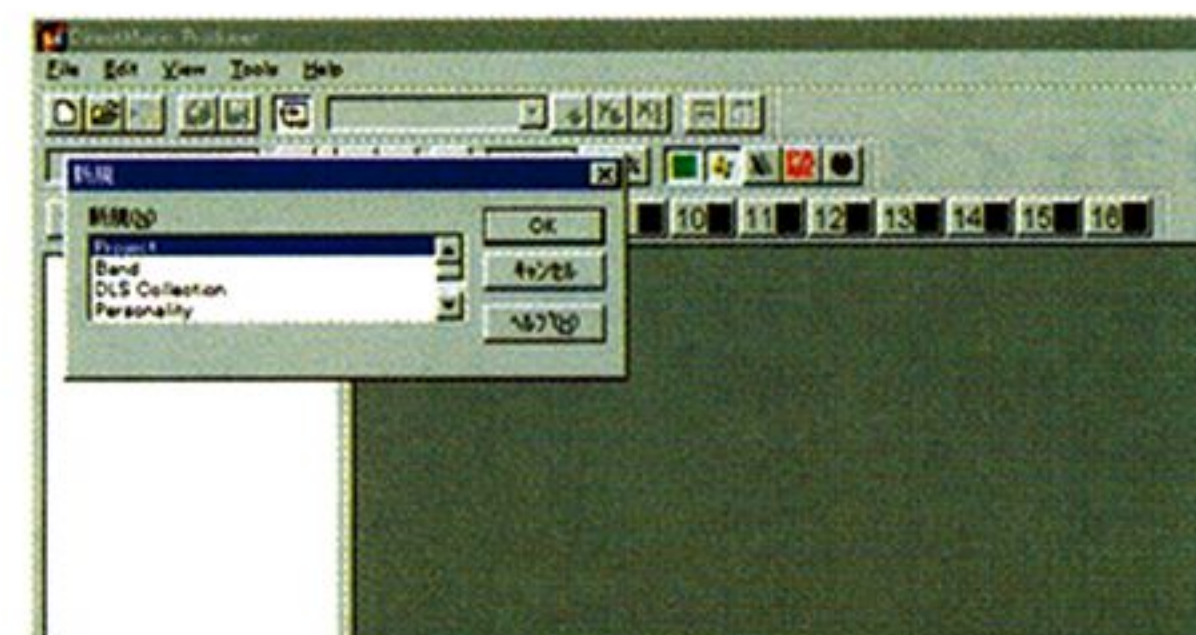



図4　新しいファイルを作成します

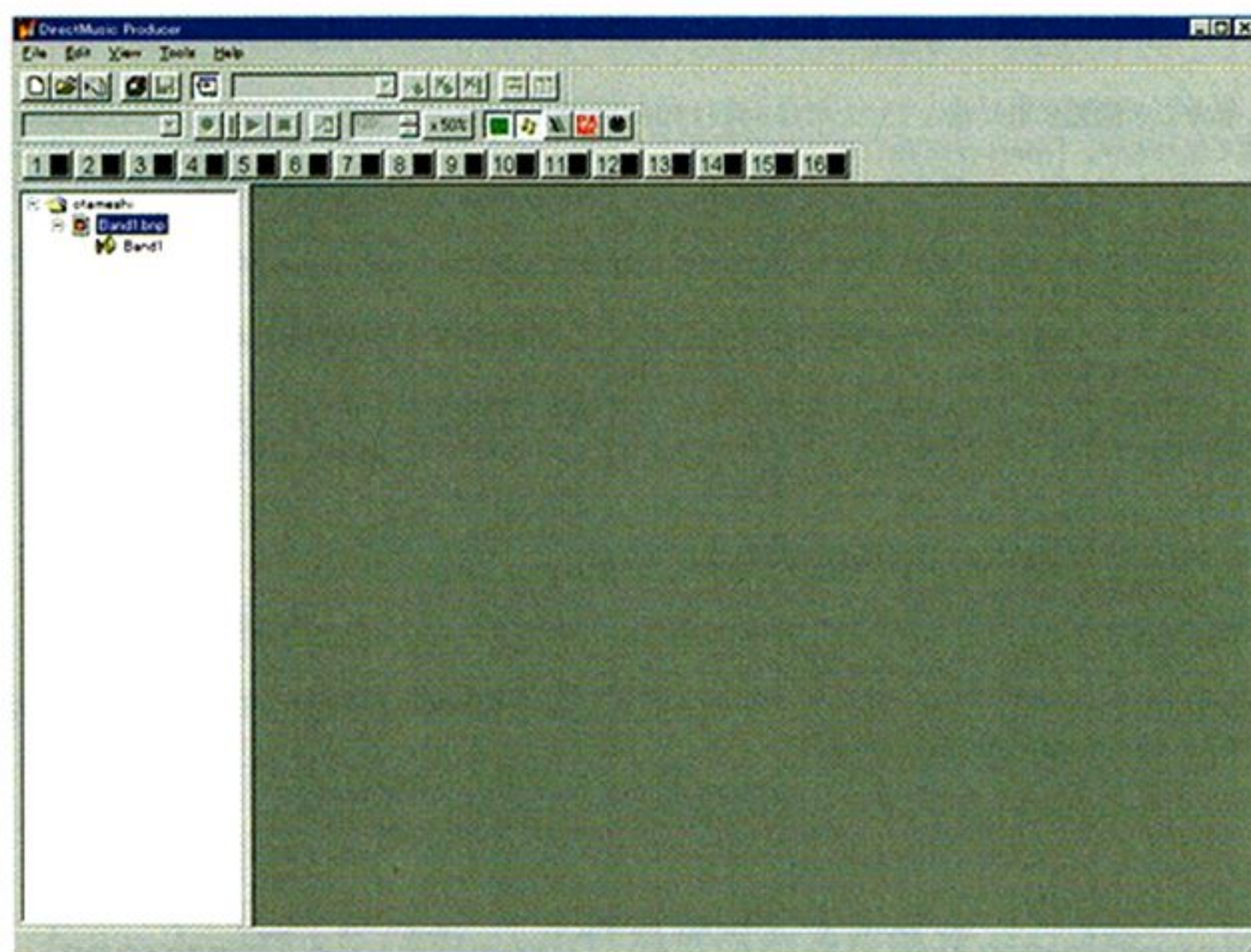



図5　プロジェクトbandを追加しました

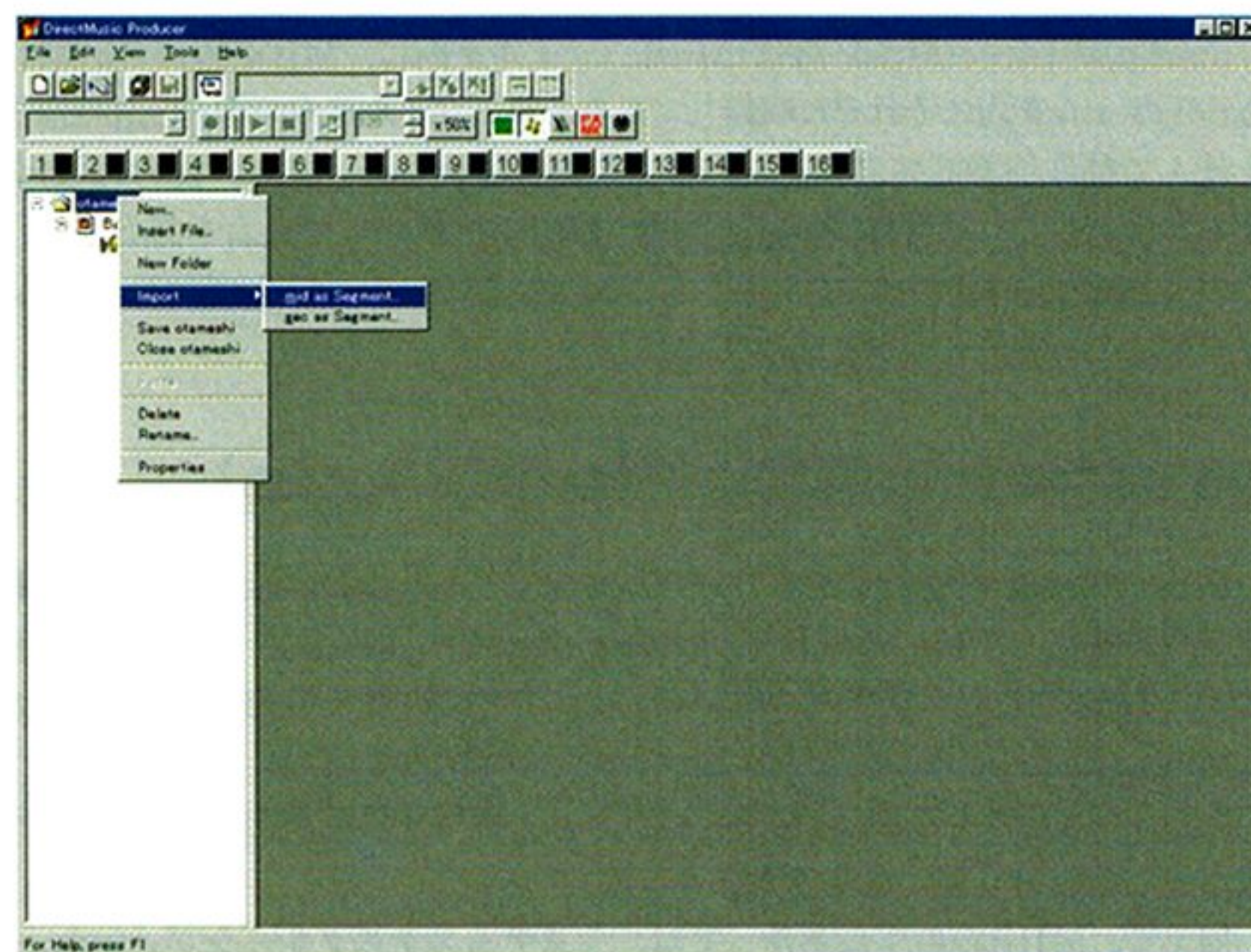



図6　メニューからMIDIファイルをインポートします

⑥ 曲をPlayした結果をもとにbandの再調整を行います。自分のイメージに近いものを作るためには何度もbandを調整，その結果をセグメントのbandトラックに貼り付けて，曲をPlayして確認という作業が続くことになると思います。
もし，どうしてもDirectMusic Producerにプリセット(最初からセット)されている楽器の音色が気に入らない場合は，DLS(DownLoadable Sounds)を研究してみてください。DirectMusicはDLS Level1を完全にサポートしているそうなので，「GM？　そんなのまだ使ってんの？　オレのサウンドはカスタム楽器＆サンプリング満載のチョー迫力で……(以下省略)」という人は速攻でDLSを試してみてくださいデス……。とりあえず「mari-p」はプリセットされている音色のみで調整してみました。
⑦ 最終的な調整が完了したら，File-Runtime Save As，またはRuntime Save All Filesを選択してファイルを書き出します(図11)。目的のファイルはプロジェクトファイルがあるフォルダのひとつ下のRuntime Filesというフォルダ内にあります。DirectMusicではこのファイルがBGMを再生(Play)するのためのデータということになります。

以上，手順をざっと解説してきましたが，これはDirectMusic Producerの使い方のほんの一部にすぎません。
DirectMusic ProducerにはDirectMusicが提供する新しい，さまざまな表現のパターンを編集するためのいろいろな使い方があります。私自身，とても使いこなせているとはいえない状態なので残念ながら詳しい説明はできませんが(ドキュメントが英文なので読み進めていくのには時間と忍耐が必要です)，DirectMusicの新しい表現に興味のある方は，ぜひ！　気合で挑戦してほしいと思います。
それから，DirectMusic Producerで曲の調整を始めると気がつくと思いますが，1つひとつの音はモノラルで再生されます(当然DLSもモノラル)。そのため，どうしてもステレオで……という場合にはいろいろと細工・工夫をしなければなりません。SDKのDirectMusicサンプル曲のような音を作り出すにはたくさんの試行錯誤が必要みたいです。みなさんも，がんばってみてください！

使用するインタフェイスの概要

DirectMusicには新しい，さまざまな要素が盛り込まれていますが，「mari12.exe」では単純にBGM再生をMIDI StreamからDirectMusicに置き換えてみることだけをやってみました。
ドキュメントによれば，DirectMusicは「completely COM-based(完全COM準拠)」とのことなので必要なインタフェイスは主に「CoCreateInstance」によって獲得します(COMについては注2参照)。
ということで当然ながらDirectDrawやDirectSoundのように，最初のインタフェイスを獲得するための専用の関数はありません……そのための専用のインポートライブラリのリンクも必要なくなりました。
とりあえず「mari-p」では，次のインタフェイスおよびメソッドを使っています。

インタフェイス：IDirectMusicLoader
　メソッド：：SetSearchDirectory
　メソッド：：GetObject

インタフェイス：IDirectMusicPerformance
　メソッド：：Init
　メソッド：：AddPort
　メソッド：：PlaySegment
　メソッド：：IsPlaying
　メソッド：：Stop
　メソッド：：CloseDown

インタフェイス：IDirectMusicSegment
　メソッド：：SetRepeats

＊共通メソッド：：Release

(1) IDirectMusicLoader

インタフェイス「IDirectMusicLoader」を提供するローダオブジェクトは主にDirectMusicで使われるファイルからオブジェクトを実体化する作業を受け持ちます。
ここでは先にDirectMusic Producerで作成したセグメントデータ(拡張子sgtのファイル)からIPersistStream(注3参照)を介してセグメントオブジェクトを実体化する作業を担当しています。
2つのメソッドはいずれもその作業に関係するものです。
メソッド「SetSearchDirectory」はファイル(ここでは目的のセグメントファイル)が存在しているディレクトリ(パス)をセットし，もうひとつのメソッド「GetObject」はファイル名からオブジェクト(ここではセグメントオブジェクト)を実体化，オブジェクトのインタフェイスポインタを獲得します。

(2) IDirectMusicPerformance

インタフェイス「IDirectMusicPerformance」を提供するパフォーマンスオブジェクトは総合的な演奏(再生)の管理を担当します。
ここではセグメントを演奏するために，まずメソッド「Init」で初期化処理

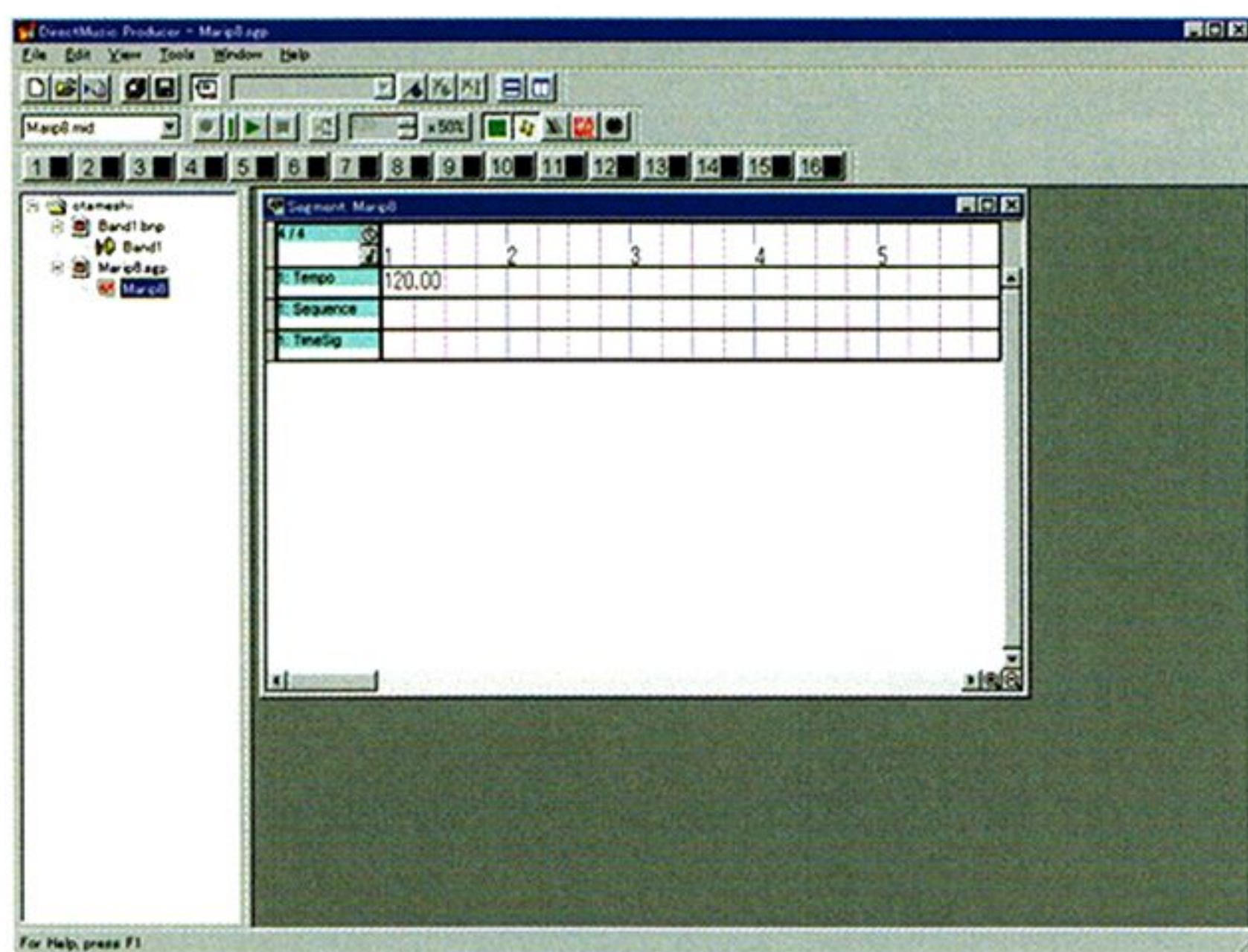



図7　インポートしたMIDIファイルの内容を表示します

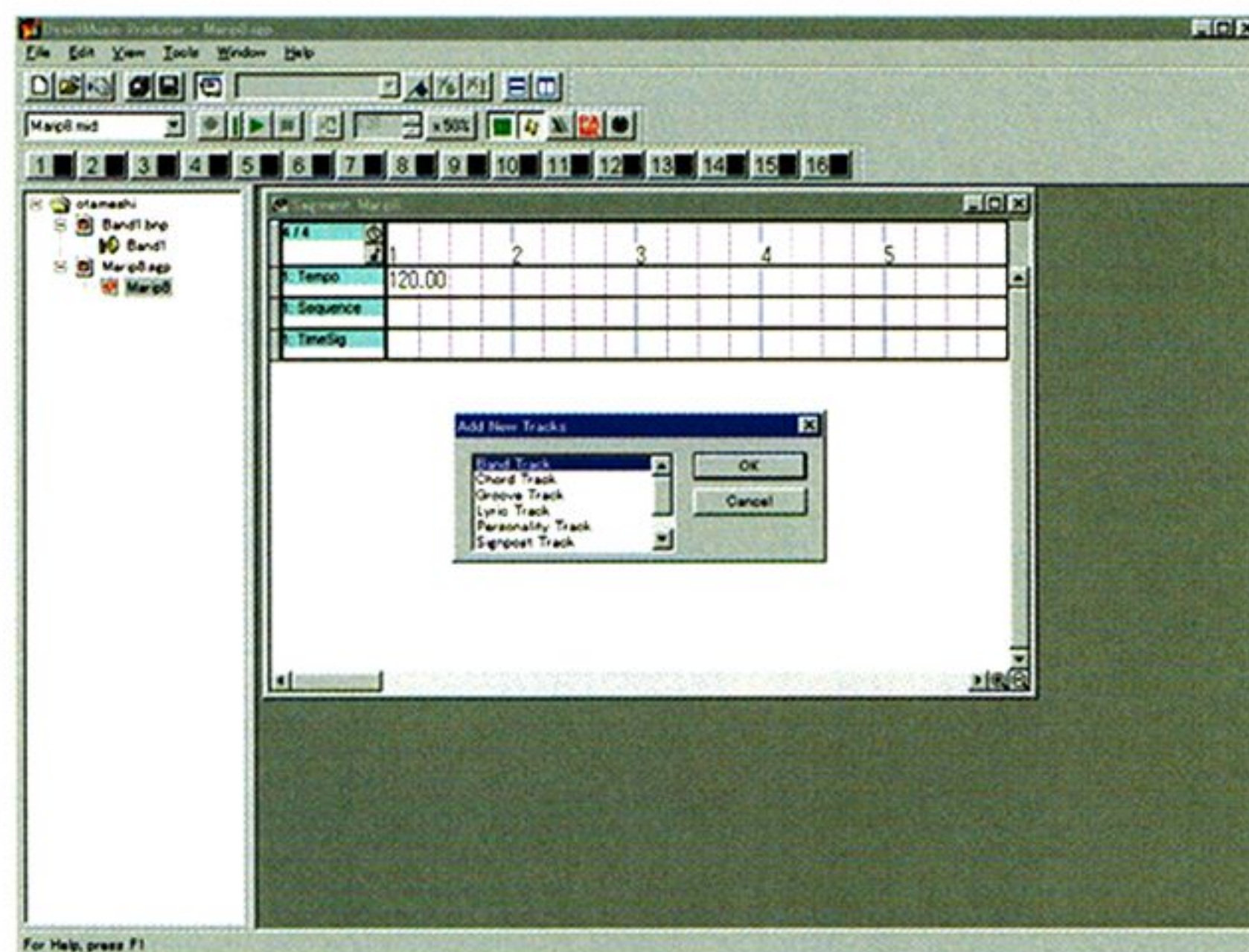



図8　bandトラックを追加します

を行い，「AddPort」でポートの割り当てを行います。

「PlaySegment」はその名前が示すようにセグメントの演奏を開始し，「Stop」は停止します。メソッド「IsPlaying」はセグメントが演奏中かどうかの確認をするためのものです……「PlayPri」サンプル内では使用されていませんが，「mari-p」ではエラーが発生した場合にセグメントのクリーンアップ処理をする際の確認用として，またはユーザーの急なアプリケーション終了処理に対処するために使用しています。メソッド「CloseDown」はオブジェクトをクリーンアップする前に終了のための処理を行います。

(3) IDirectMusicSegment

インタフェイス「IDirectMusicSegment」を提供するセグメントオブジェクトはひとつ以上のトラックを構成するミュージックデータブロックとそれにアクセスするためのメソッドで構成されています。

セグメントの初期化および実体化，そしてインタフェイスの獲得はローダーオブジェクトが，演奏(再生)の管理はパフォーマンスオブジェクトがやってくれるので必要なメソッド「SetRepeats」はリピート回数をセットするものだけとなっています。

ちなみにこのオブジェクトのインタフェイスを「CoCreateInstance」で獲得することもできますが，その場合はローダの「GetObject」が処理してくれる仕事を，こちら側で1つひとつ手作業でやっていかなければなりません。普通，その必要はないのでオブジェクトの生成，およびインタフェイスの獲得はローダに任せます。

注2
COM(Component Object Model)はマイクロソフトがバイナリベース(コンパイル＆リンクが完了したEXEやDLL)でプログラムの部品化・共用化を実現するための規格・仕様規定です。詳しく知りたい方には次の書籍が役立ちます。
「Inside COM」：Dale Rogerson著，アスキー出版局

注3
IPersist……で始まるインタフェイスを実装しているオブジェクトは「永続オブジェクト」と呼ばれ，記憶媒体に自己の状態をセーブ/ロードできます。DirectMusicのローダはIPersistStreamを介して目的のオブジェクトに対し，ファイルの内容に基づいて自身を初期化するように指示を出すものと思われます。
IPersist……で始まる構造化記憶・永続オブジェクトに関する概要をつかみたい方には次の書籍が役に立ちます。
「UNDERSTANDING ActiveX AND OLE」：David Chappell著，アスキー出版局

コーディング時のポイント

では，BGM再生機能を実装するため，実際にDirectMusicのインタフェイス/メソッドをコーディングする手順を見てみましょう。コーディングにあたって参照しているSDKのサンプルコードは「PlayPri」フォルダ内の「playpri.cpp」などです。なお，本レポートではCOMコンポーネントの使い方についての説明や，各メソッドごとの細かい説明は省略します。

COMについての説明はそれだけで別のレポートが書けるくらいになってしまいますし，メソッドの詳細解説はSDKヘルプの翻訳とほとんど同じになってしまいます……(誤解を招かないような翻訳ができる自信もありません)。

もし，COMコンポーネントをプログラム上で使用した経験のない方，もしくはDirectXコンポーネントを使ったプログラミングの経験はあるが「CoInitialize」は知らない，という方は注2で紹介している書籍の必要箇所，またはほかのCOMについて解説している書籍を読んでから，取り組んでください。

また，各メソッドについては前のセクション「使用するインタフェイスの概要」をもとにSDKのサンプルコード「playpri.cpp」を解読すれば，おおかたつかめるのではないかと思います。それに英文とはいえ，コードを解読できる人には引数などの詳細はDirectX SDKのヘルプを見るほうが確実なのでは，とも思います。できれば，このセクションを読む前に「PlayPri」フォルダ内のコードに目を通して，実際の処理がどのようなものであるのか確認しておいてください。

前置きが長くなりましたが，「mari-p」での処理の流れはだいたい次のようになっています。

(1) ゲームスタート時の処理

- 「CoInitialize」の呼び出し
- 「CoCreateInstance」で「IDirectMusicLoader」のポインタGet！
- 「CoCreateInstance」で「IDirectMusicPerformance」のポインタGet！
- パフォーマンスオブジェクトの初期化処理・ポートの割り当て
- ローダによるセグメントのロード・実体化，「IDirectMusicSegment」のポインタGet！

↓

(2) ゲーム中の処理

- (必要に応じて)セグメントのロード・実体化，「IDirectMusicSegment」のポインタGet！
- (必要に応じて)リピート回数などの設定
- セグメントの演奏(再生)開始
- セグメントの演奏(再生)停止
- (必要に応じて)セグメントのクリーンアップ

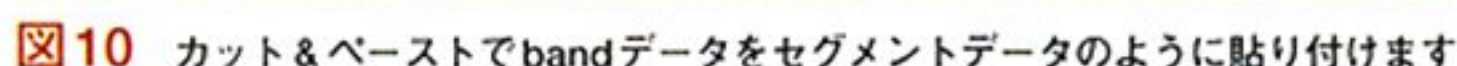
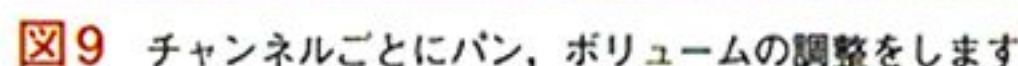
↓

(3) ゲーム終了時の処理

- (必要に応じて)セグメントの演奏(再生)停止
- セグメントのクリーンアップ
- パフォーマンスオブジェクトの終了処理・クリーンアップ
- ローダオブジェクトのクリーンアップ

図9 チャンネルごとにパン，ボリュームの調整をします

図10 カット＆ペーストでbandデータをセグメントデータのように貼り付けます

・「CoUninitialize」の呼び出し

上記の処理をコーディングしてゆく作業において注意が必要となるのは，ほとんどがローダに関するものになります。サンプルコード「playpri.cpp」でもローダに関連するもの以外の部分は，それほど読みにくくないのではないかと思います。というわけで，以下にそのポイントを書いてみます。

ローダに関連する処理のポイント

①メソッド「SetSearchDirectory」が不要な状況

サンプルコード「playpri.cpp」ではセグメントファイルの存在するパスをセットするためにちょっと面倒な処理をしていますが，これは状況によっては不要な処理です……というのは，データとして使用するセグメントファイルは一般的に，実行ファイルと同じディレクトリにあるのが自然な状態であるからです。

デフォルトではローダはカレントのディレクトリからセグメントなど目的のファイルを探します。

そのため，もしカレントのディレクトリに必要となるファイルが存在していることが確実な場合には，メソッド「SetSearchDirectory」の呼び出しは省略可能です。

その際は「DMUS_OBJECTDESC」構造体のメンバ，「wsFileName」にも名前だけ(パス不要で)セットすればよいとのことです(当然ながら「SetSearchDirectory」を使用してパスをセットした場合も「wsFileName」にはファイル名だけでOK)。

②WCHAR

ドキュメント，またはサンプルコードを読んでいくとすぐに気がつくと思いますが，ローダの2つのメソッド「SetSearchDirectory」「GetObject」に必要となるパス/ファイル名はワイドキャラクタ(WCHAR)のポインタまたは配列になっています。

プログラム内でDirect Music以外の処理のために「char」配列などにパス名/ファイル名をセットしている場合は，「mbstowcs」などの関数で使う前に必ず変換しておいてください。

③マクロ定義

ローダで使用する「DMUS_OBJECTDESC」構造体のメンバの配列ではいくつかのマクロが使われています。

たとえばWCHAR型の配列，「wsFileName」の定数式は「DMUS_MAX_FILENAME」というマクロ名になっていますが，これはヘッダ(dmusici.h)内で「MAX_PATH」と同じという定義がされていて，「MAX_PATH」は「windef.h」で260と定義されています。実際の値が必要なときには必ずヘッダファイルをチェックしてください。

④セグメントファイルをリソースからロードしたい

ゲームのプログラミングをやっている方なら，配布する際のゲームの関係ファイル数はなるべく少なくして，データ類はリソースに一括してEXE/DLLに組み込んでおく……という方法が一般的ではないかと思います。

が，メソッド「GetObject」に渡さなくてはならないのはあくまで「ファイル名」なので，リソースからセグメントデータを読み込ませるには一工夫必要です。ところでDirectMusic版「mari-p」のファイル構成をご覧になりましたか？

ゲームを構成しているファイルに拡張子「sgt」のついているものはありません。「mari-p」ではセグメントファイルをリソースとして構成していて，ゲーム起動時にリソースからメモリにデータを読み込み，そこから一時処理用のファイルをハードディスクに作成して「GetObject」に渡し，セグメントの実体化が完了したらファイルを削除する……という方法を取っています。でも，これはちょっと芸のない方法です(自分でいうのもなんですが)。

もし研究する時間のある方はカスタムローダ(既存のローダオブジェクトを使わない)でセグメントオブジェクトを立ち上げる方法をチェックしてみてください。セグメントオブジェクトは「IPersistStream」を提供しているので，もっと効率のよい方法が必ずあります。

以上，私の経験から注意すべきポイントについて書いてみましたが，DirectMusicでBGMを再生するためのコードはMIDI Streamで同じことをするためのものよりずっと簡単です。さぁ！　ガンガン，コードを打ち込んで，目的のアプリケーションをビルドしてしまいましょう！……と，ここで私の場合は思わぬエラーにぶちあたってしまいました。

DirectMusicはコンパイラを選ぶ？

いきなり関係のない話ですが，私が普段使っている開発環境はインプライズの「BorlandC++5.01」です。初めてWindowsのプログラムに挑戦したときから，大切なパートナーとしてずっと使っています。次々と新しい開発環境がリリースされていきますが，その魅力は少しも衰えていません。

その「BorlandC++5.01」がDirectMusic関連のコードをコンパイル中にいきなりエラーを出しました。エラーなんて別に驚くことではないのですが，問題はエラーの発生した場所にあります。そのエラーは「dmusici.h」からで，そこには次のような記述がありました。

```
__int64
```

「Visual C++5.0」のドキュメントによれば，__int64は64ビットの整数の

図11　できあがったらファイルを書き出します

型定義でマイクロソフト固有の仕様の中にあります。

私にこの__int64の存在を初めて教えてくれたのは，insideWindows(ソフトバンクの雑誌です)の6月号(1998年)に掲載されていた「WDJ Selection」というコーナーの「Visual C++の__int64とストリームI/O」という記事でした。そこには「Visual C++5.0」のストリームI/Oでの__int64の使用に関する問題点について書いてありましたが，その記事(正確には__int64)が私に与えた印象はそれほど大きなものではなく，今回のエラーにぶつかるまでは，その存在もすっかり忘れておりました。

今回，急いで記事の内容をチェックしてみると，「C++Builder」「WatcomC/C++11.0」は__int64を正しくサポートしているとありましたが，私の使っているコンパイラ，「Borland C++5.01」の名前はありませんでした……。いくつもの困難な課題を共に乗り越えてきた「Borland C++5.01」でしたが，今回ばかりはどうにもならないんじゃないか……という感じでした。「まだまだ十分現役でいけるのになぁ」などとぼやいてみましたが，そんなセリフをいくら口にしたところで状況が好転するわけでもありません。しかたがないので，速攻で「Visual C++5.0」を立ち上げてビルドの作業を開始しました。

Direct Musicのコードをコンパイル＆リンクするには__int64をサポートしている開発環境が必要になるみたいです。

おわりに

最後に今回のレポートの総括として，DirectMusicの長所·短所を簡単にまとめてみたいと思います。

●長所

①ハードウェアに依存しない音作りができる

DirectMusicコンポーネントを使うことで，アプリケーションは音楽演奏(再生)を行う際の環境の違いについて，対応する必要がなくなりました。

演奏される音は，とりあえずみな同じに聞こえます(それがよい音かどうかはまた別の問題ですが)。

②プログラムのコード量を減らせる

mari-pでBGMを実装するためのコード量を，DirectMusic版/MIDI Stream版でそれぞれ単純に比較したところ，DirectMusicを使用した場合はMIDI Streamを使用したときに比べて，3分の1強は減らせる……という結果になりました。すごいことだと思います。

③マルチメディア関連固有の処理に精通していなくてもコードが組める

たとえば「mmio……」など，低水準アクセスのための知識は不要になりました。

●短所

①データ量が増える

これはしかたのないことかもしれませんが，一般的にセグメントファイルはMIDI Streamのファイルより大きくなってしまいます。

②再生される１つひとつの音はモノラルである

もしかしたら将来のバージョンで改良されるのかもしれません……期待しています。

ほかにもあるかもしれませんが，私は自身試してみた結果として，「DirectMusicの長所はその短所を補ってあまりある！」という結論になりました。

そしてほかのDirectXコンポーネントの機能改良過程を振り返ってみると，DirectMusicの未来もまた明るいな……という感じがします。

DirectMusicによって実現された新しい表現の可能性は，クリエイターにとっては新しいアイデアを生み出す源にもなることでしょう。

いままでなかったといわれるようなワクワクドキドキするゲームが登場してくる日をとても楽しみにしています。

では，また次回の投稿作品でお会いしましょう！

図12　おまけ：ゲームmari-pのヘルプ画面

図13　おまけ：MIDI strramプレイヤー

x86アセンブラ講座 MMX命令を使ってみよう

影山裕昭　Kageyama Hiroaki

MMX命令はまだコンパイラが直接コードを出してくるようなものではありませんから，マシン語レベルのプログラミングでなくては効率的に扱えません。想定されている用途が特殊なものなのですが，このようなレベルで処理するのがアセンブラプログラミングの醍醐味といえるでしょう。

MMXテクノロジは，従来のPentiumプロセッサにSIMD（Single Instruction Multiple Data）技術を導入し，一度に複数のデータを同時処理することを可能にしました。MMXテクノロジは，8ビットや16ビットデータに対する演算処理を繰り返し行う画像，オーディオ，ビデオ，音声処理の速度を高速化することを目的に開発されたものです。

MMXテクノロジを最初に実装したMMX Pentiumは1997年1月に発表されました。その後発表されたインテルのPentium II，Celeronはもちろん，AMDのK6，K6-2，IDTのWinChip 2，CyrixのM IIなど，現在流通しているほとんどのCPUは，MMXテクノロジを実装しています。

データ型

MMXテクノロジは，図1に示すように4つの新しい64ビットデータ型を処理します。8つのバイト，4つのワード，2つのダブルワードを64ビットデータに格納するデータ型は，パックドデータと呼びます。クワッドワードはパックドデータのひとつの型として取り扱うことができます。たとえば，パックドバイト同士の加算では，同時に8つのバイトデータの加算をすることができます。

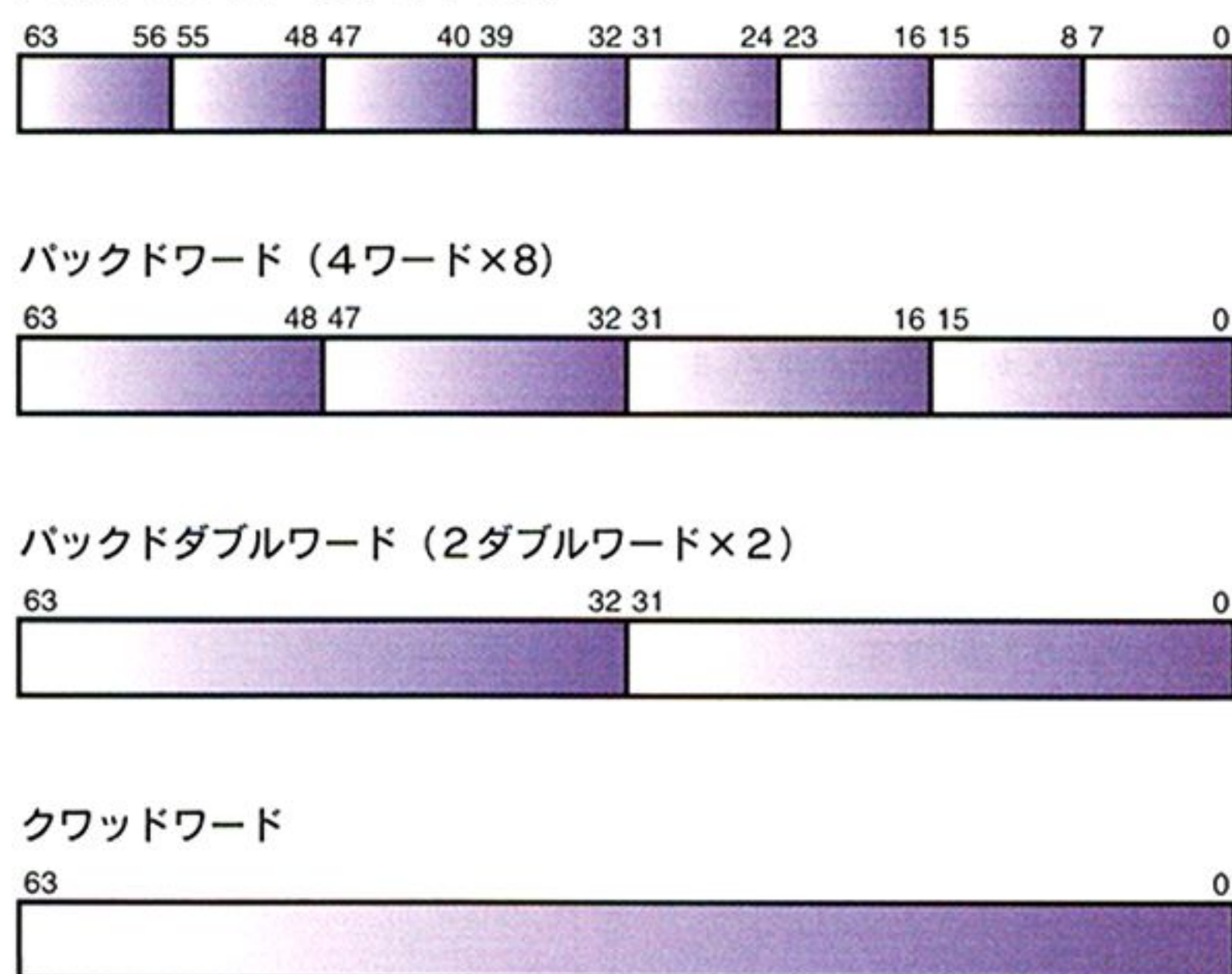
8ビットデータが主体の画像処理や，16ビットデータが主体の音声処理では大幅に効率化されることが予想できます。従来は画像の合成などがあると，24ビットデータからRGBを8ビットずつ取り出し，64ビットレジスタでいちいち演算していました。レジスタの1/8しか使っていなかったものが，きっちり64ビット使って8個のデータを同時処理できれば飛躍的な性能向上が見込めます。

パックドバイト（8バイト×8）
63　56 55　48 47　40 39　32 31　24 23　16 15　8 7　0

パックドワード（4ワード×8）
63　48 47　32 31　16 15　0

パックドダブルワード（2ダブルワード×2）
63　32 31　0

クワッドワード
63　0

図1　MMXでサポートされているデータタイプ

MMXレジスタ

MMXテクノロジを搭載したプロセッサは，8つの64ビット汎用レジスタを備えています（図2）。レジスタ名称はMM0～MM7で，これらは浮動小数点レジスタ上に別名定義されています。MMXレジスタは64ビットのパックドデータを保持することができます。

フラグ変化について

もともとのPentiumプロセッサに備わっている加減算命令（add，sub）や比較命令（cmp）は，演算結果によりフラグレジスタ内のSFフラグの状態を変化させますが，MMX命令セットの演算結果はSFフラグの内容に影響を与えません。したがってMMX命令セットの演算結果を利用して条件分岐をすることはできません。しかしながら，MMX命令にはPCMPEQ，PCMPGTという比較命令が用意されています。これらの命令については後述します。

飽和演算（サチレーション）

通常，加減算の結果がデータの表現範囲を超えた場合，ラップアラウンド（丸め）された値が結果として残ります。つまり正の表現範囲を超えたり，負の表現範囲を超える（オーバーフロー）のとき，最上位ビットを切り捨てます。符号なし1バイトデータのラップアラウンドの例を示します。

```
0xFF + 0x03 = 0x02
0x02 - 0x04 = 0xFE
```

ラップアラウンドされると都合が悪い場面もあります。たとえば24ビッ

63　0
MM7
MM6
MM5
MM4
MM3
MM2
MM1
MM0

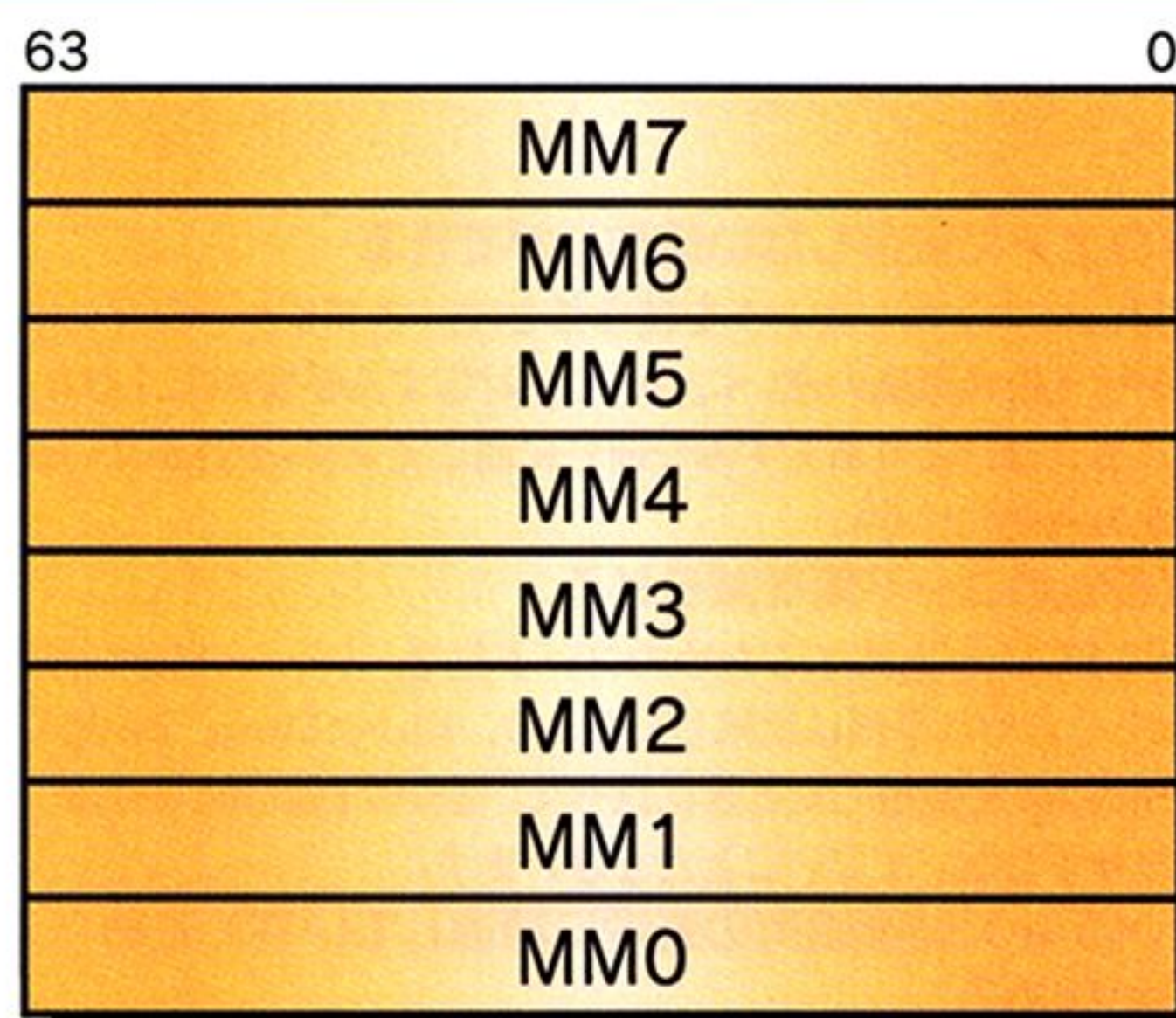
図2　MMXレジスタ

	下限		上限	
	16進	10進	16進	10進
符号付きバイト	0x80	−128	0x7f	127
符号なしバイト	0x00	0	0xff	255
符号付きワード	0x8000	−32768	0x7fff	32767
符号なしワード	0x0000	0	0xffff	65535

図3

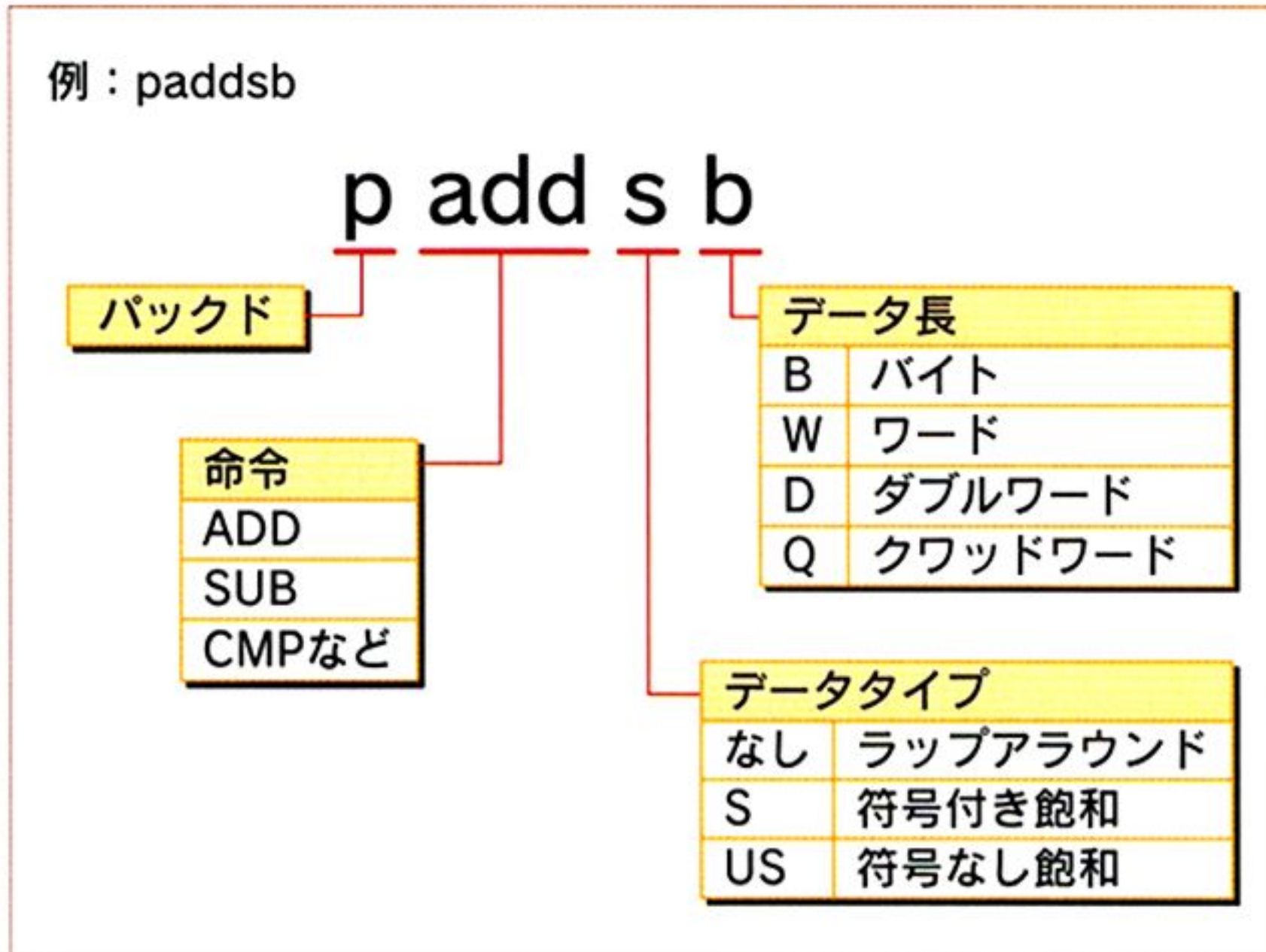



図4

トのビットマップデータのピクセル輝度はRGBそれぞれについて8ビットの符号なし整数で0～255の範囲にあります。輝度の加算減算の結果，最高輝度(255)を超える場合は最高輝度のままであってほしいですし，最低輝度(0)を下回る場合は最低輝度のままであってもらいたいものです。MMXテクノロジの飽和演算を使えば，表現範囲の下限上限値を保持します。先ほどの演算を符号なし飽和演算で行うと，

```
0xFF + 0x03 = 0xFF
0x02 - 0x04 = 0x00
```

となります。

MMX命令セット

MMXテクノロジでは新しく57個の命令が追加されました。これらはEMMS命令を除いたすべての命令が，頭文字Pで始まります。ほとんどの命令は図4に示すような規則に従っています。たとえばpaddsb命令は"パックドバイト符号付き飽和加算"とわかります。

● パックド加算および減算

パックド加算およびパックド減算命令は，処理するデータタイプ，データ長によって複数用意されています。各命令のソースオペランドにはMMXレジスタ/64ビットのメモリ内容(表記中mm/m64)を指定し，デスティネーションオペランドにはMMXレジスタ(表記中mm)を指定します。各命令はソースオペランドとデスティネーションオペランドを加算減算し，データタイプに従ってラップアラウンド，または飽和させて，デスティネーションオペランドに格納します。

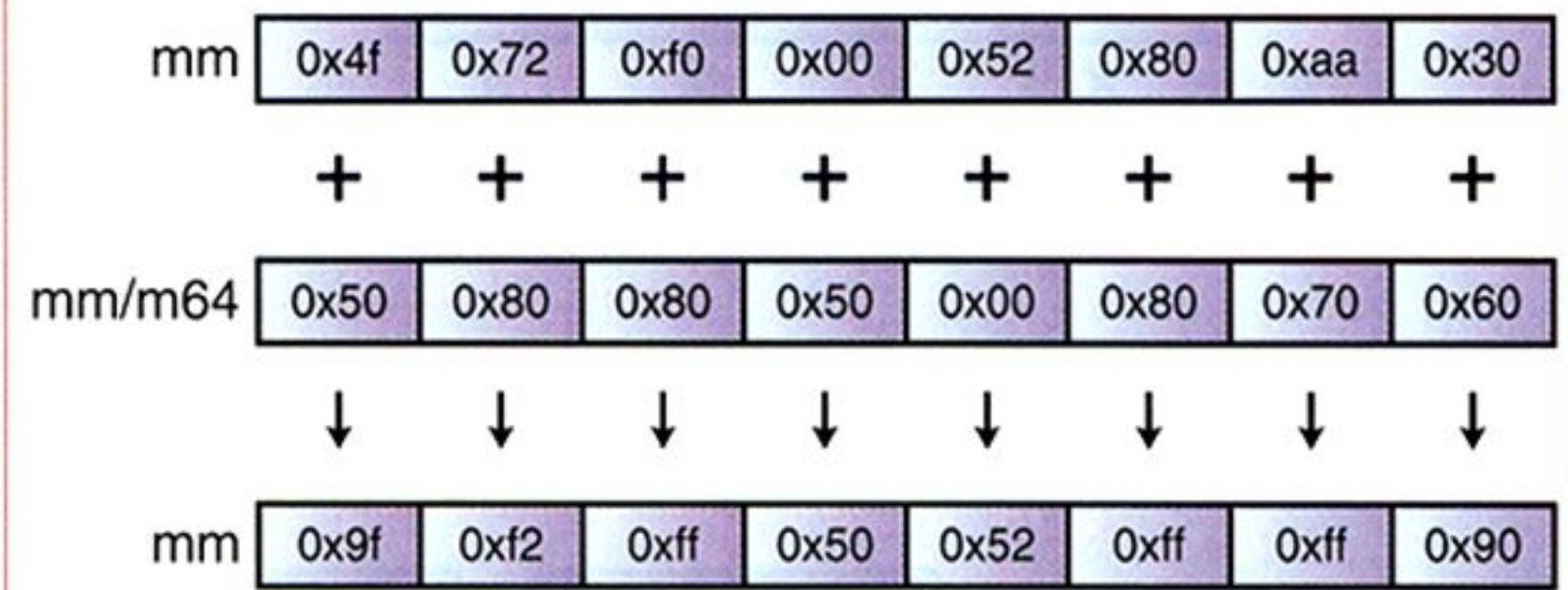
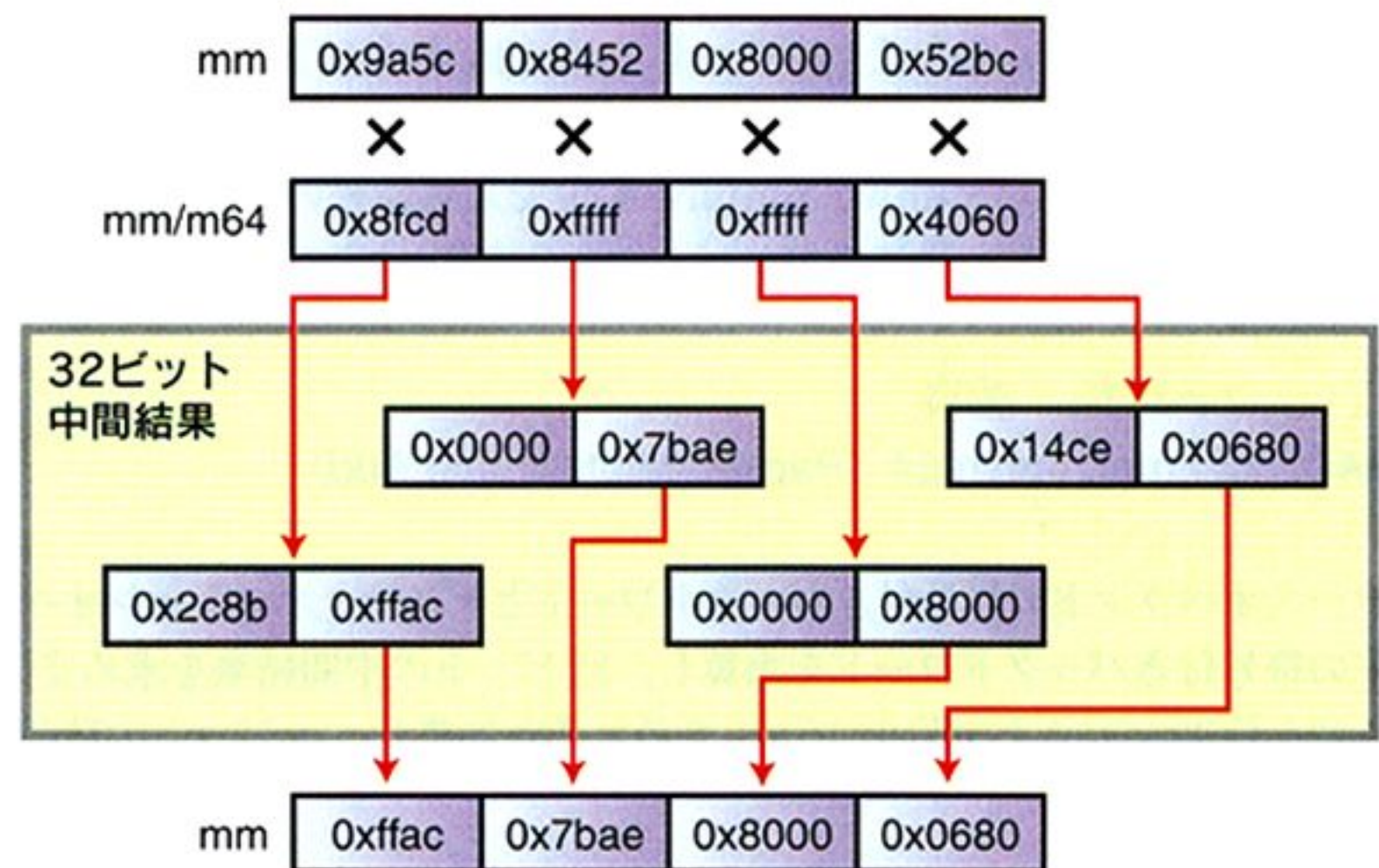



図5

図6

● パックド乗算

PMULHW mm, mm/m64 (Packed Multiply High)
PMULLW mm, mm/m64 (Packed Multiply Low)

ソースオペランドのパックドワードデータ要素とデスティネーションオペランドのパックドワードデータ要素を乗算します。32ビットの中間結果を

表1 パックド加算および減算命令

	ラップアラウンド Packed Add/Sub	符号付き飽和 Packed Add/ Sub with Saturation	符号なし飽和 Packed Add/ SubUnsigned with Saturation
パックドバイト加算	PADDB mm,mm/m64	PADDSB mm,mm/m64	PADDUSB mm,mm/m64
パックドワード加算	PADDW mm,mm/m64	PADDSW mm,mm/m64	PADDUSW mm,mm/m64
パックドダブルワード加算	PADDD mm,mm/m64	――	――
パックドバイト減算	PSUBB mm,mm/m64	PSUBSB mm,mm/m64	PSUBUSB mm,mm/m64
パックドワード減算	PSUBW mm,mm/m64	PSUBSW mm,mm/m64	PSUBUSW mm,mm/m64
パックドダブルワード減算	PSUBD mm,mm/m64	――	――

PMADDWD mm,mm/m64（パックド乗算・加算）

mm: 0x8000 0x0001 0x8000 0x8000
× × × ×
mm/m64: 0x0002 0x002 0x8000 0x8000

0xffff 0x0000 + 0x0000 0x0002

0x4000 0x0000 + 0x4000 0x0000

mm: 0xffff0002 0x80000000

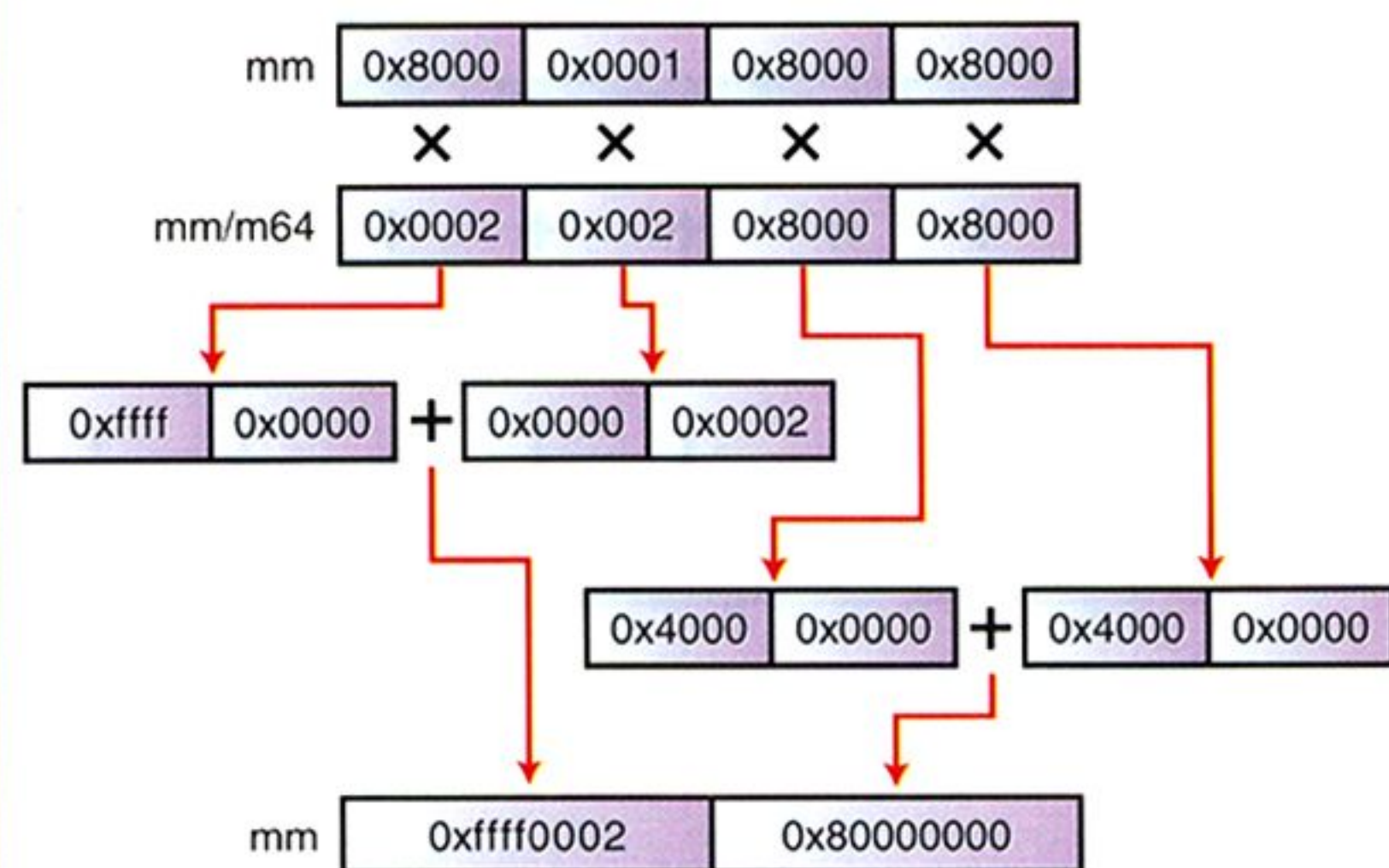



図7

求めたあと，pmulhw命令は上位ワード，pmullw命令は下位ワードをデスティネーションオペランドに格納します。32ビットの乗算結果が必要な場合は，同一データをpmulhw，pmullw命令で乗算結果の上位ワードと下位ワードを求めたあと，後述のアンパック命令を使ってマージします。

● パックド乗算・加算

PMADDWD mm, mm/m64（Packed Multiply and Add）

ソースオペランドの符号付きパックドワードとデスティネーションオペランドの符号付きパックドワードを乗算し，32ビットの中間結果を求めます。さらに上位32ビットと下位32ビットをそれぞれ加算し，32ビットの積和をデスティネーションオペランドに格納します。ソースとデスティネーション4つのワード全部が0x8000のときのみ，演算結果が0x80000000にラップアラウンドします。

● 比較命令

ソースオペランドの符号付きパックドデータとデスティネーションオペランドの符号付きパックドデータの等号比較または大小比較をします。比較結果が等号またはデスティネーション＞ソースのとき，対応するデスティネーションオペランドのパックドデータの全ビットを1にセットします。比較結果が偽の場合は，対応するデスティネーションオペランドのパックドデータの全ビットを0にセットします。

● 変換命令

パック命令はソースオペランドのパックドデータと，デスティネーションオペランドのパックドデータをマージし飽和させてデスティネーションオペランドに書き込みます。たとえばPACKSSWB命令では，変換前の符号付きワード値が，符号付きバイト値の下限の0x80より小さい場合は0x80に飽和し，上限の0x7fより大きければ0x7fに飽和します。

アンパック命令は，ソースオペランドの上位または下位データ要素と，デスティネーションオペランドの上位または下位データ要素を，インタリーブ（交互にアクセス）してデスティネーションオペランドに書き込みます。ソースオペランドがすべて0の場合には，デスティネーションオペランドの下位または上位要素を，ゼロ拡張する結果になります。

● 論理命令

64ビットデータのビット操作を行います。

● シフト命令

デスティネーションオペランドの内容をソースオペランドで指定したビッ

表2　比較命令

命　令	説　明
PCMPEQB mm,mm/m64	パックドバイト等号比較 (Packed Compare for Equal)
PCMPEQW mm,mm/m64	パックドワード等号比較 （〃）
PCMPEQD mm,mm/m64	パックドダブルワード等号比較 （〃）
PCMPGTB mm,mm/m64	パックドバイト大小比較 (Packed Compare for Greater Than)
PCMPGTW mm,mm/m64	パックドワード大小比較 （〃）
PCMPGTD mm,mm/m64	パックドダブルワード大小比較 （〃）

表3　パック命令

命　令	説　明
PACKSSWB mm,mm/m64	符号付きパックドワード → 符号付きパックドバイト（符号付きワードの値が，符号付きバイトの表現範囲を超える場合，飽和させる）
PACKUSWB mm,mm/m64	符号付きパックドワード → 符号なしパックドバイト（符号付きワードの値が，符号なしバイトの表現範囲を超える場合，飽和させる）
PACKSSDW mm,mm/m64	符号付きパックドダブルワード → 符号付きパックドワード（符号付きダブルワードの値が，符号付きワードの表現範囲を超える場合，飽和させる）

PCMPEQW mm, mm/m64
（パックドワード等号比較）

mm: 0x0000 0x2000 0x5000 0x4000
== == == ==
mm/m64: 0x1000 0x2000 0x3000 0x4000
↓ ↓ ↓ ↓
mm: 0x0000 0xffff 0x0000 0xffff

図8

PACKSSWB mm,mm/m64
（符号付きパックドワード→符号付きパックドバイト）

mm/mm64: 0x1234 0x0004 0x00f0 0x8000
mm: 0x004f 0xffff 0x0050 0x0000

mm: 0x7f 0x04 0x7f 0x80 0x4f 0xff 0x50 0x00

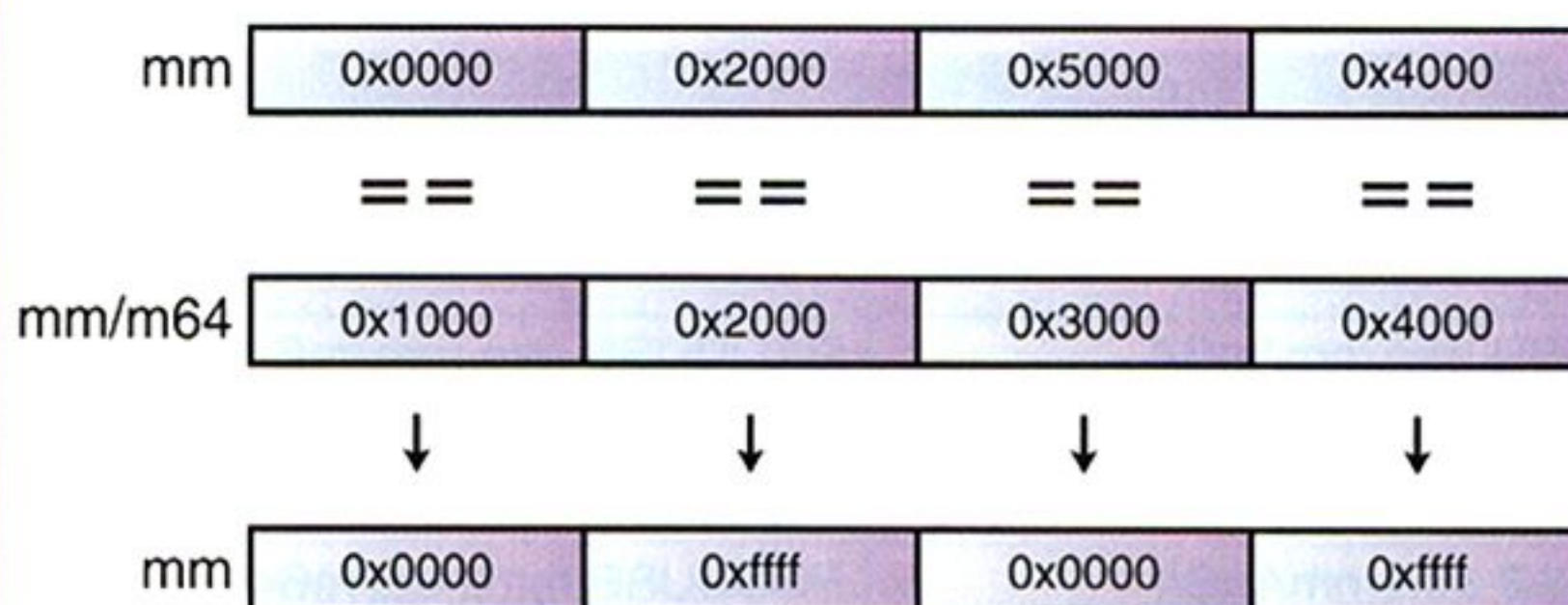
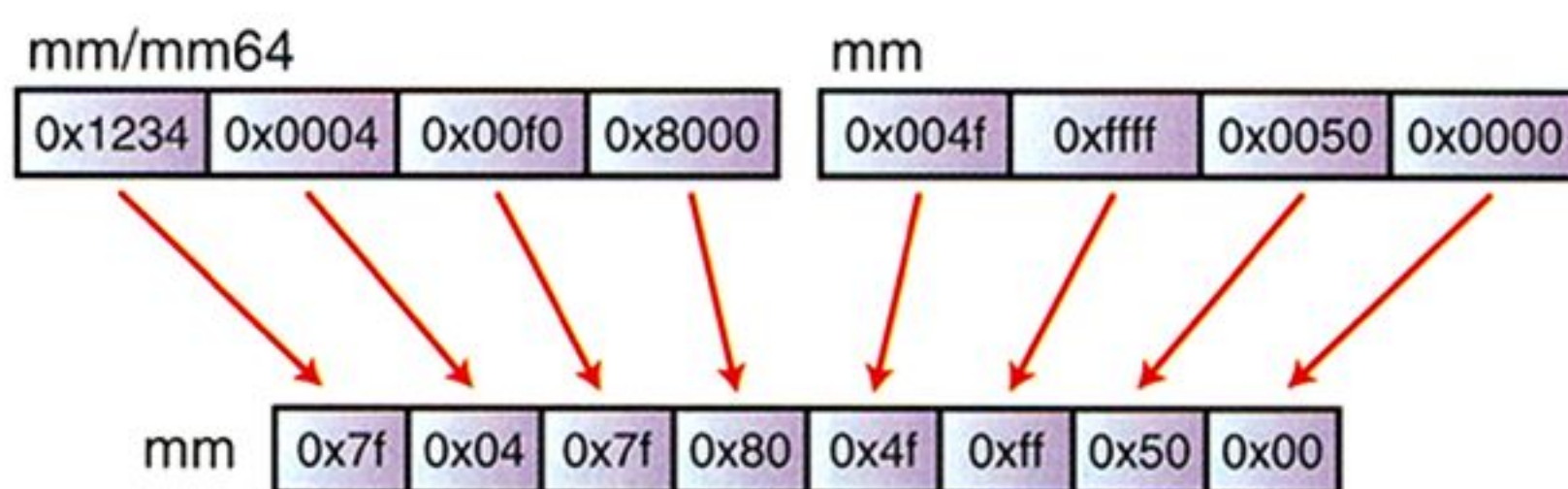



図9

ト数だけ左または右にシフトします。MMX命令セットでシフト命令だけが，ソースオペランドに即値を指定することができます。論理シフトまたは算術シフトをすることができます。論理左(右)シフトで空いたビットには0がセットされます。算術右シフトでは空いたビットは符号ビットと同じ値をセットします。

● データ転送命令

データ転送命令には，32ビットデータ転送と64ビットデータ転送があります。MOVD命令は32ビットデータの転送をするもので，MMXレジスタと整数レジスタ/メモリ(表記中m32/r)間の32ビット幅のデータ転送をします。整数レジスタ/メモリの内容をMMXレジスタに転送する場合，MMXレジスタの上位32ビットには0がセットされます。

64ビットデータ転送にはMOVQ命令を使います。MMXレジスタとMMXレジスタ/メモリ間の64ビット幅のデータ転送をします。

● MMXステートクリア命令 EMMS（Empty MMX State）

EMMS命令はMMX命令使用ルーチンの最後，または浮動小数点演算を使用する可能性のあるルーチンを呼び出す前に，必ず実行する必要があります。前述のとおり，MMXレジスタは浮動小数点レジスタに別名定義されています。MMX命令使用後，EMMS命令の実行以前に浮動小数点命令を実行すると，浮動小数点スタックオーバーフローが発生する可能性があります。これは浮動小数点例外を発生するか，正しくない結果を導く原因になります。

● 実行クロックとペアリング

すべてのMMX命令の実行スループットは1です。これは1クロックサイクルごとにMMX命令を実行できるということです。ただし，乗算命令についてはレイテンシ＝3のため，乗算結果が使えるようになるのは乗算命令実行を含めて3クロック後になります。

MMX Pentiumプロセッサでは，MMX命令をUパイプ，Vパイプでペ

表4　アンパック命令

命　令	説　明
PUNPCKHBW mm,mm/m32	上位4バイトをインタリーブしてパックドバイトを生成
PUNPCKHWD mm,mm/m32	上位2ワードをインタリーブしてパックドワードを生成
PUNPCKHDQ mm,mm/m32	上位ダブルワードをインタリーブしてパックドダブルワードを生成
PUNPCKLBW mm,mm/m32	下位4バイトをインタリーブしてパックドバイトを生成
PUNPCKLWD mm,mm/m32	下位2ワードをインタリーブしてパックドワードを生成
PUNPCKLDQ mm,mm/m32	下位ダブルワードをインタリーブしてパックドダブルワードを生成

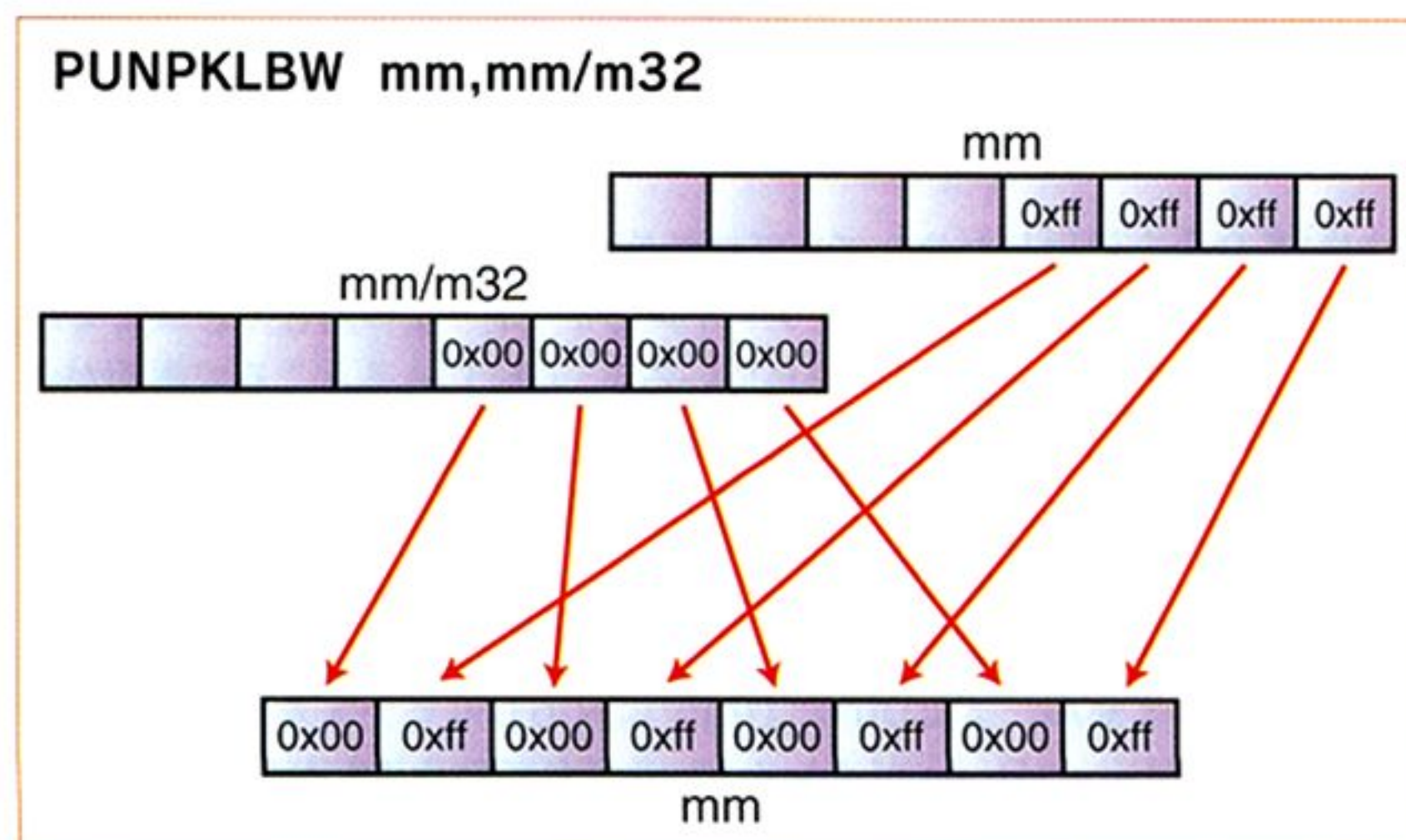



図10

表5　論理命令

命　令	説　明
PAND mm,mm/m64 (Bitwise Logical And)	デスティネーションオペランドの64ビットと，ソースオペランドのANDを取る。対応する両方のビットが1なら1に，さもなければ0をデスティネーションオペランドに書き込む
PANDN mm,mm/m64 (Bitwise Logical And Not)	デスティネーションオペランドの64ビットを反転した結果と，ソースオペランドのANDを取る。対応する両方のビットが1なら1に，さもなければ0をデスティネーションオペランドに書き込む
POR mm,mm/m64 (Bitwise Logical Or)	デスティネーションオペランドの64ビットと，ソースオペランドのORを取る。対応する両方のビットが0なら0に，さもなければ1をデスティネーションオペランドに書き込む
PXOR mm,mm/m64 (Bitwise Logical Exclusive Or)	デスティネーションオペランドの64ビットと，ソースオペランドのXORを取る。対応する両方のビットが異なる値なら1に，さもなければ0をデスティネーションオペランドに書き込む

表6　シフト命令

命　令	説　明
PSLLW mm,mm/m64 PSLLW mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドワードを左に論理シフトします
PSLLD mm,mm/m64 PSLLD mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドダブルワードを左に論理シフトします
PSLLQ mm,mm/m64 PSLLQ mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのクワッドワードを左に論理シフトします
PSRLW mm,mm/m64 PSRLW mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドワードを右に論理シフトします
PSRLD mm,mm/m64 PSRLD mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドダブルワードを右に論理シフトします
PSRLQ mm,mm/m64 PSRLQ mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのクワッドワードを右に論理シフトします
PSRAW mm,mm/m64 PSRAW mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドワードを右に算術シフトします
PSRAD mm,mm/m64 PSRAD mm,imm8	デスティネーションオペランドで指定するビット数だけ，ソースオペランドのパックドダブルワードを右に算術シフトします

表7　データ転送命令

命　令	説　明
MOVD mm,r/m32	ソースオペランドの整数レジスタまたは32ビットメモリロケーションから，デスティネーションオペランドのMMXレジスタへ，32ビット幅のデータ転送をします。MMXレジスタの上位32ビットには0をセットします
MOVD r/m32, mm	ソースオペランドのMMXレジスタの下位32ビットから，デスティネーションオペランドの整数レジスタまたは32ビットメモリロケーションへ，32ビット幅のデータ転送をします
MOVQ mm,mm/m64	ソースオペランドのMMXレジスタまたは64ビットメモリロケーションから，デスティネーションオペランドのMMXレジスタへ，64ビット幅のデータ転送をします
MOVQ mm/m64,mm	ソースオペランドのMMXレジスタから，デスティネーションオペランドのMMXレジスタまたは64ビットメモリロケーションへ，64ビット幅のデータ転送をします

アリングすることができます。乗算ユニットとシフトユニットはそれぞれひとつしか持たないため，連続する乗算命令や，シフト/パック/アンパック命令はペアリングできません。また，整数レジスタ，メモリに対するアクセスは，Uパイプでのみ実行可能となっています。その他の命令はU，Vどちらのパイプでも実行できます。

● コーディング技法

すべてのMMX命令を紹介しました。次に，実際のプログラミングで使う機会の多いと思われるMMX命令のコーディング例をいくつか紹介します。

MMXレジスタに即値をセットする

整数レジスタに値を設定するにはMOV命令を使いますが，MMX命令のMOVQ命令はデスティネーションオペランドに即値を指定することができません。即値を設定するときは，メモリ上に定数値を確保しておき，64ビットのメモリロケーションをMMXレジスタに書き込みます。

```
DWORD TEISUU[2] = { 0x12345678, 0x12345678 };
_asm
{
   movq  mm0, TEISUU
}
```

MMXレジスタに0をセットする

```
pxor  mm0, mm0
```

MMXレジスタの全ビットに1をセットする

```
pcmpeqw  mm0, mm0
```

MMXレジスタのパックドデータに1をセットする

```
pxor        mm0, mm0
pcmpeqw  mm1, mm1
psubw      mm0, mm1
```

MMXレジスタのパックドワード(パックドダブルワード)に符号付き2n－1をセットする

```
pcmpeqw  mm0, mm0
psrlw      mm0, 16-n (psrld mm0, 16-n)
```

MMXレジスタのパックドワード(パックドダブルワード)に符号付き－2nをセットする

```
pcmpeqw  mm0, mm0
psllw      mm0, n (pslld mm0, n)
```

符号なしパックドバイトをゼロ拡張する

入力：mm0 符号なしパックドバイト
　　　mm7 0
出力：mm0 下位4バイトをゼロ拡張したパックドワード
　　　mm1 上位4バイトをゼロ拡張したパックドワード

```
pxor          mm7, mm7
movq         mm1, mm0
punpcklbw  mm0, mm7
punpckhbw  mm1, mm7
```

ここではmm0の符号なしパックドバイトを符号なしパックドワードにゼロ拡張しています。結果として，mm0にはmm0の下位4バイトからゼロ拡張した4つのパックドワード，mm1にはmm0の上位4バイトからゼロ拡張した4つのパックドワードを書き込みます。

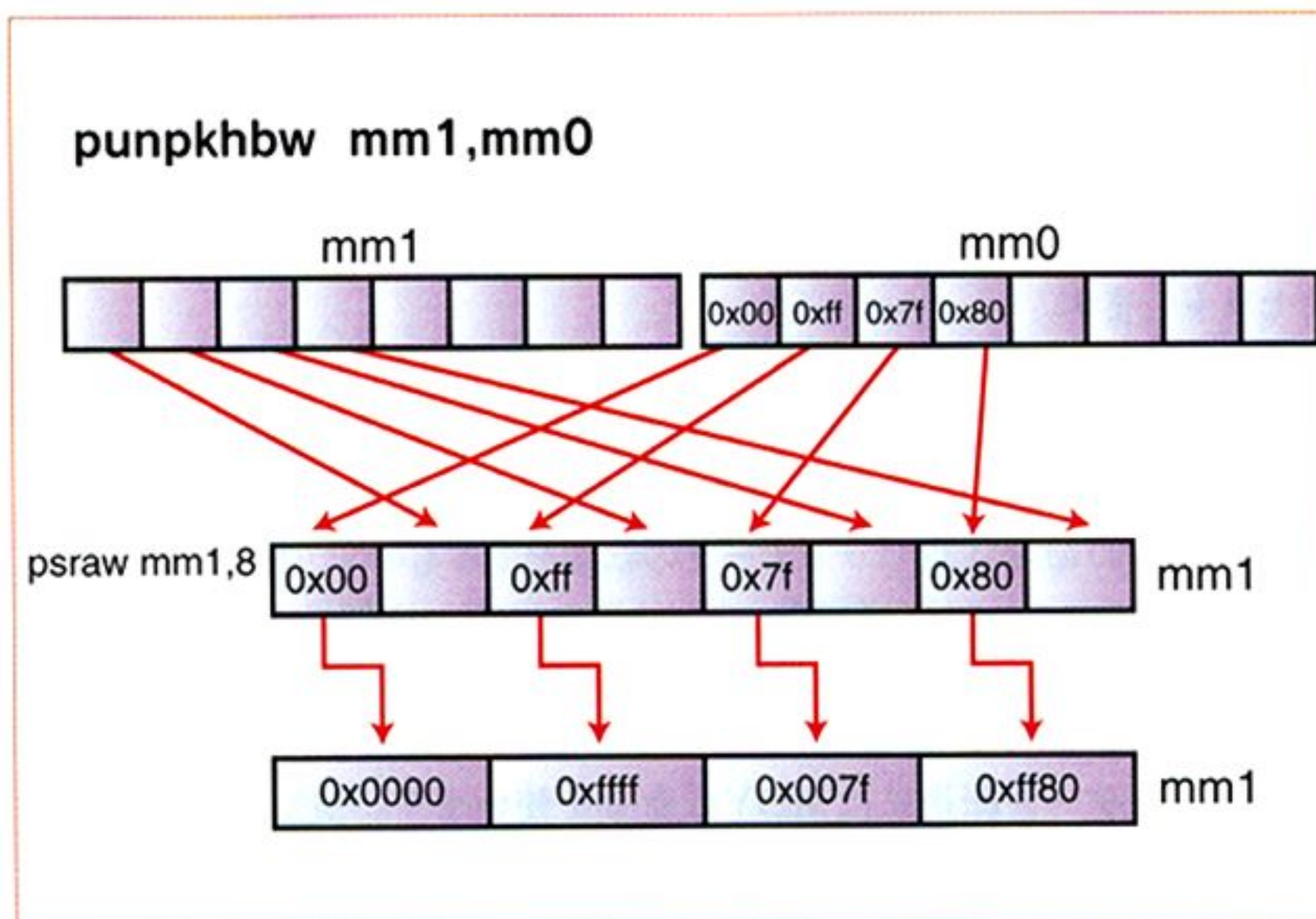



図11 mm0の上位4バイトを符号拡張

符号付きパックドバイトを符号拡張する

例ではmm0の符号付きパックドバイト上位4バイトをmm1，下位4バイトをmm0に符号拡張してパックドダブルワードに変換します。符号ビット保持のために算術シフトを使います。

入力：mm0 符号付きパックドバイト
出力：mm0 下位4バイトを符号拡張したパックドワード
　　　mm1 上位4バイトを符号拡張したパックドワード

```
punpkhbw mm1, mm0
punpklbw  mm0, mm0
psraw      mm0, 8
psraw      mm1, 8
```

符号なし数値の差の絶対値を求める

psubusb mm0，mm1，psubusb mm1，mm0の2つの符号なし飽和減算は，結果が負の場合は0に飽和します。2つの減算結果をPORすれば正の演算結果がmm0に書き込まれるわけです。この手法はパックドワードには使えますが，パックドダブルワードには飽和減算がないため使えません。

入力：mm0 符号なしパックドバイト
　　　mm1 符号なしパックドバイト
出力：mm0 符号なしオペランドの差の絶対値

```
movq      mm7, mm0
psubusb  mm0, mm1
psubusb  mm1, mm7
por         mm0, mm1
```

サンプルプログラム紹介

MMX命令を使用したサンプルプログラムを作成しました。24ビットカラーで保存された2つのBitmapファイルをα：1－αの割合で合成して画面に表示します。テレビドラマを見ていると，CMに入るところで画面がホワイトアウトする場面がありますよね。画像合成を使えば，そのような効果を演出することができます。

最初にプログラムを実行するPCがMMX命令に対応しているか調べます。MMX命令を実装していないCPUでは，サンプルプログラムを実行することができません。またDirectDrawに対応したビデオカードと，Windows95/98にDirectDrawがインストールされていることが必須です。

sample.exeを実行すると，sample.exeと同じディレクトリにある，"sample_a.bmp"，"sample_b.bmp"を読み込みます。これらは320×240

の24ビットの画像データでBitmap形式で保存されていれば，お手持ちの画像ファイルをリネームして表示することもできます。Visual C++をお持ちであれば，ご自身でソースプログラムを変更してみて，いろいろと遊んでみてください（ワークスペースはsample.dswです）。ところで申しわけありませんが，プログラムは途中で終了することができませんので悪しからず。画面に変化がなくなったらESCキーを押して終了させてください。

さて，CPUがMMXテクノロジに対応しているか判定するには，eaxレジスタに1をセットしてCPUID命令を実行します。その結果edxレジスタにセットされる値のbit23が1であれば，MMX命令を実装しています。CPUID命令はPentiumプロセッサになって搭載された命令であるため，本来はCPUID命令自体の存在も調べる必要があるのですが，ここでは省略しています。

```
//MMX命令の実装を調べる
        int     check_mmx;
        _asm{
                mov         eax,1
                cpuid
                mov         check_mmx,edx
        }
        //bit23が0ならMMX命令を実装していない
        if (check_mmx & 0x800000 == 0)
        {
//MMXテクノロジが実装されていない場合の処理
        }
```

前号の『マシン語ってなぁに？』では，画像にモーションブラー効果をかけるプログラムを作成しました。ピクセルのGet，Put処理をC++で記述し，プライマリサーフェイスに対して直接処理していました。そうすると1ピクセル処理するごとに，プライマリサーフェイスのLock，Unlock処理をすることになります。Lock，Unlockはたいへん遅い命令であり，実行速度を低下させる原因になっていました。

今回はプライマリサーフェイスへの直接描画をやめて，すべて仮想画面に対して処理を行うようにしました。仮想画面に対する書き込みが終わったら，仮想画面の内容をバックサーフェイスに転送するようにしています。そうしてからDirectDrawのFlipを使って，プライマリサーフェイスとバックサーフェイスを切り替えています。

画像合成のアルゴリズムは，アルファブレンディングを使用しています。あるピクセルに注目したとき，qにpのピクセル輝度を0～100％の割合で合成したsのピクセル輝度は以下の数式で表されます。

$$s(r, g, b) = \alpha \times p(r, g, b) + (1-\alpha) \times q(r, g, b)$$

αは$0 \leqq \alpha \leqq 1$で指定します。r，g，bはピクセルごとの赤，緑，青の輝度を表し，24/32ビットカラーでは各々8ビット（0～255）で表現します。この数式を変形すると，

$$s = \alpha \times p + (1-\alpha) \times q$$
$$s = \alpha \times p + q - \alpha \times q$$
$$s = \alpha \times (p-q) + q$$

さらにαを小数で扱うのは面倒なので，αを128倍したものと（p－q）を乗算し，そのあとで128で除算して辻褄をあわせます。

$$s = \alpha \times (p-q) \div 128 + q \quad (0 \leqq \alpha \leqq 128)$$

途中，$\alpha \times (p-q)$の演算結果が8ビットに収まらないので，最初からr，g，bのピクセル輝度をパックドワードデータ形式に変換しています。以降はすべての処理をワードデータで行い，最後にパック命令を使ってバイトデータに戻しています。途中にある除算命令はMMX命令にありません

図12　画像1

図13　画像2

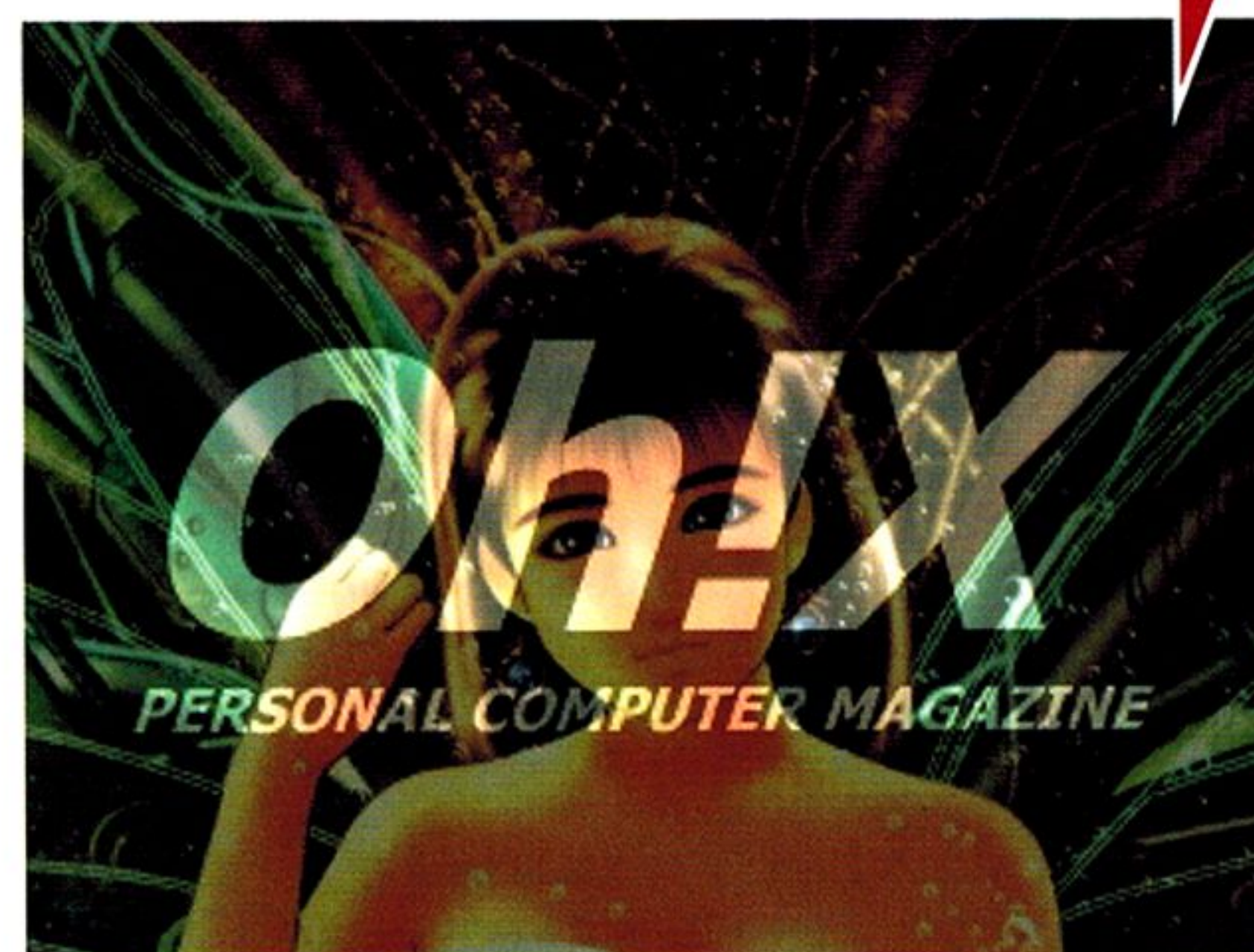



図14　MMX命令でブレンド

リスト

```
//ビットマップ画面の次ラインへのオフセットを計算
BitmapPitch = (Src->GetWidth() - dxx) * 4;
//仮想画面の次ラインへのオフセットを計算
WorkPitch = (640 - dxx) * 4;
//ビットマップ画面の先頭アドレスを取得
BitmapAdrs = (BYTE*)Src->GetPtr() + sy * Src->GetWidth() + sx * 4;
//仮想画面の先頭アドレスを取得
WorkAdrs = (BYTE*)Dest->GetStartAdrs() + desty * 4 * 640 + destx * 4;
//ビットマップ画面から仮想画面へ転送

_asm {
        pxor        mm0,mm0
        movd        mm1,alpha
        punpcklwd   mm1,mm1
        punpckldq   mm1,mm1
        mov         ecx,dxx
        mov         ebx,dyy
        mov         esi,BitmapAdrs
        mov         edi,WorkAdrs
        mov         eax,ecx
L1:     movd        mm3,[esi]
L2:     movd        mm4,[edi]
        punpcklbw   mm3,mm0
        add         esi,4
        add         edi,4
        punpcklbw   mm4,mm0
        psubsw      mm3,mm4
        psllw       mm4,7
        pmullw      mm3,mm1
        paddw       mm4,mm3
        psrlw       mm4,7
        movd        mm3,[esi]
        packuswb    mm4,mm0
        movd        [edi-4],mm4
        dec         ecx
        jne         L2
        add         esi,BitmapPitch
        add         edi,WorkPitch
        mov         ecx,eax
        dec         ebx
        jne         L1
        emms
}
```

図15　mm0の上位4バイトを符号拡張

が，除数が2 nの除算についてはシフト命令で代用できますので問題ありません。なお，サンプルプログラムで使用している関数の仕様については，ここでは説明しません。ソースリストのコメントを参照してください。

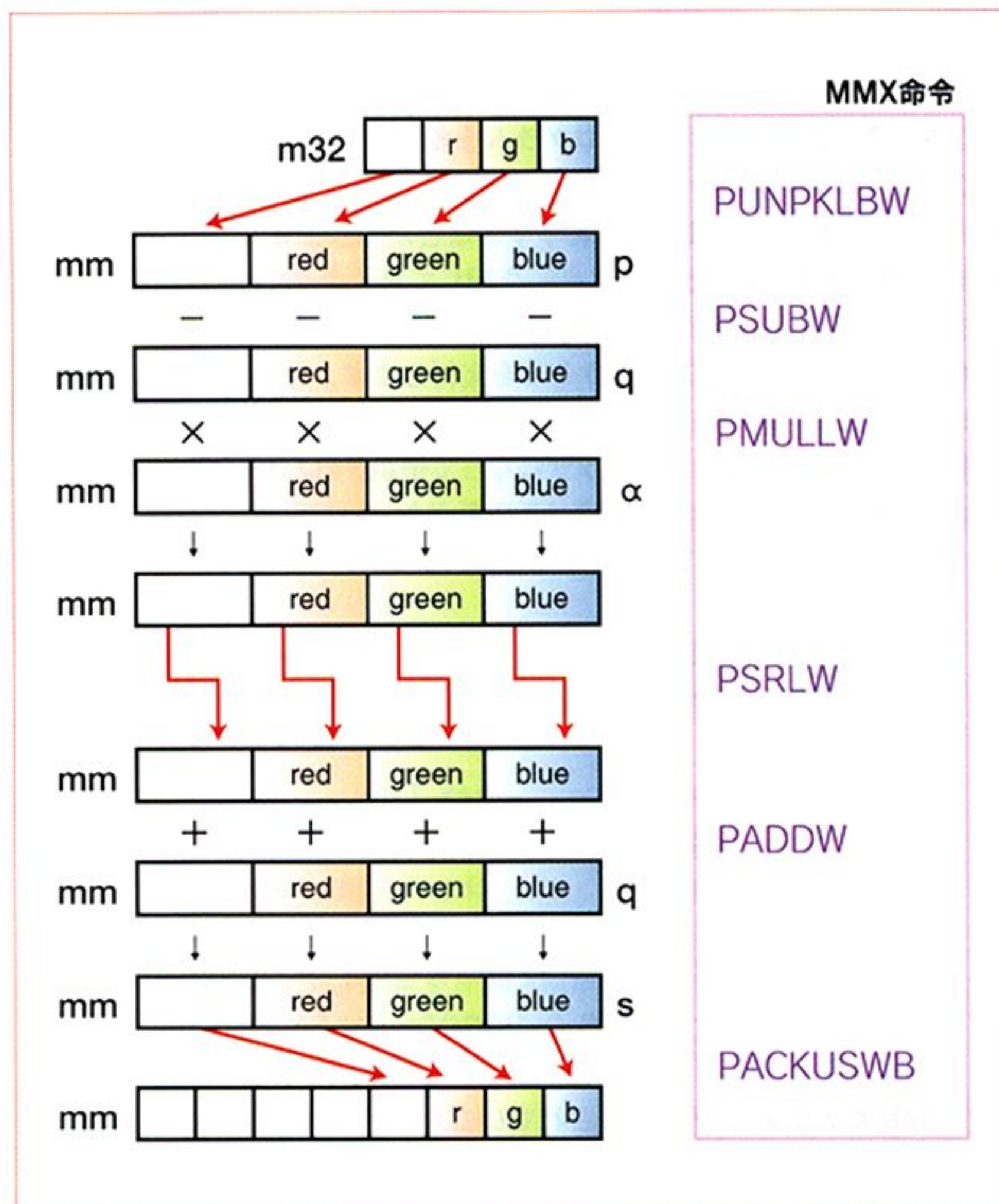
謝辞

サンプルプログラムの作成にあたっては^c氏がWebで公開されているデモプログラム〝minimal″のソースリストが大変参考になりました。快くソースプログラムの提供を承諾してくれた^c氏に，この場を借りて感謝の意を表わすとともに，氏のホームページURLを紹介いたします。インターネットに接続できる環境にある方は，ぜひアクセスしてみください。

Radium　http://cgi.din.or.jp/~keijiro/

最後に

駆け足でMMXテクノロジを紹介してきました。限られた誌面であり，説明が不十分なところもあるかと思います。MMXテクノロジについて，より理解を深めたい方は，参考文献をご覧になることをおすすめします。特に「MMXテクノロジオフィシャルガイド」は最適化について詳しく書かれています。

MMXテクノロジはソフトウェアがMMX命令に対応しないことには，その恩恵は受けられません。MMX命令は整数命令のSIMDでしたが，インテルのライバル会社であるAMDは，K6-2で苦手としていた浮動小数点演算を強化する目的で浮動小数点演算SIMD技術を導入した3D Now!を搭載しました。その後，WinChip2にも3D Now!が搭載され，今後ソフトウェアの対応が待たれるところです。3Dゲームのジオメトリ演算(オブジェクトの座標計算)では，ソフトウェアが3D Now!にネイティブ対応すれば，同クロックのPentium IIの処理能力を上回るともいわれています。

これに対抗してインテルは，1999年1月にKatmaiのコードネームで知られる次世代CPUの名称を「Pentium III」とすることを発表しました。Pentium IIIにはSSE(Streaming SIMD Extension)と呼ばれる新しいマルチメディア命令を実装しています。この命令セットは3D Now!のインテル版といわれています。

次の機会には，3D Now!命令，SSE命令に触れてみようと思います。それまでの間，私も勉強しておきますので，今までプログラミング経験のない方も，自分でなにかを作ることにチャレンジしてみてください。

参考文献

[1] MMXテクノロジオフィシャルガイド　菅原清文著　ソフトバンク刊 2,900円
[2] MMXテクノロジ最適化テクニック 小鷲英一著　アスキー出版局刊 3,500円
[3] インテルアーキテクチャMMXテクノロジ デベロッパーズガイド　インテル株式会社
資料番号：243006J-002
インテルアーキテクチャMMXテクノロジ プログラマーズリファレンス・マニュアル　インテル株式会社
資料番号：243007J-003

付録 CD-ROMの使い方

第2号も2枚組だ!

さてさて，予定外だったのですが，今回も2枚組です。CD-ROMドライブをお持ちでない方は……なんとかお友達を探してください。今回は音声トラックはありませんので，ミュージックCDプレイヤーにかけてはいけません。

付録CD-ROMはHTMLファイルビュアで閲覧することを前提に構成されています。HTMLビュアがなくてもそれぞれのディレクトリのファイルに個別にアクセスできますし，HTMLファイルはシフトJISコードで記述されていますので，テキストエディタで内容を見ることは可能ですが，なるべくHTMLビュアを調達してきてください(内容はそれほどたいしたものではありませんが)。

収録されているファイルには圧縮されているものがありますが，圧縮展開ツールは付属しておりません。別途入手してください。

それでは収録されているファイルの説明をしていきましょう。

Disk1 AzureDisc

ASM

x86アセンブラ講座のサンプルです。実行ファイルとソースコードが収録されています。

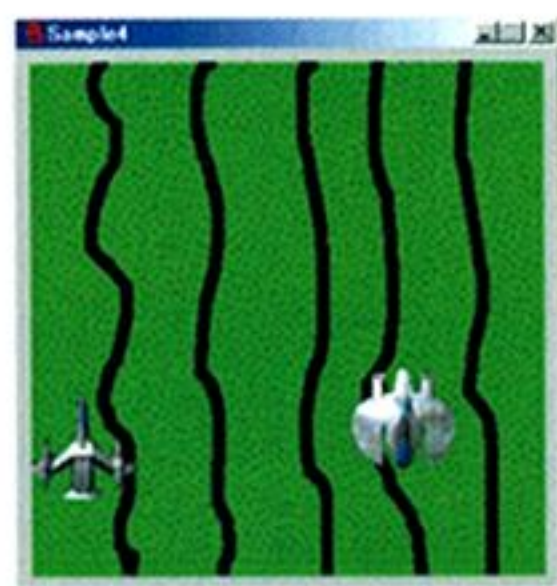
CGA

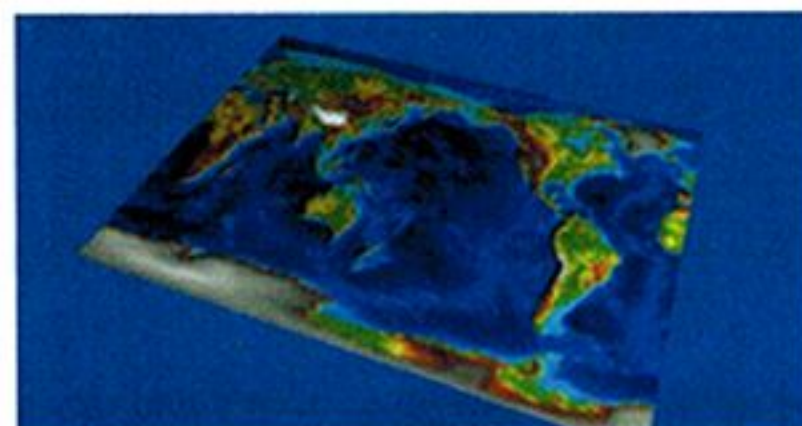
第11回アマチュアCGAコンテストの結果発表です。画像と簡単な解説，一部ムービーつきです。

CGIGAME

特集で扱ったPaerlを用いた簡単な神経衰弱ゲームです。ブラウザのBACKボタンは使用しないでください。

CPPBLDER

インプライズのC++ Builderを使った簡単なゲーム制作までの過程が収録されています。

D3D

Direct3D RM/IM解説のサンプルプログラムです。2段レンダリングであるとかバンプマッピングであるとか，これからの技術の基本となるところをやっていきます。

DINPUT

DirectInput自体は今回直接は関係のないものですが，特集記事で扱われているプログラムで，入力レイテンシのテスト用として作成されたものです。このサンプルではバッファ3で処理をしています。

DMUSIC

ゲームのBGMにDirectMusicを使うということで応募されていたプログラムです。

DPLAY

DirectPlayを使ったネットワーク多人数対戦ゲームです。最大12人でプレイできます。

F2J

仏日翻訳システムの最新版です。今度のプログラムは品詞情報などをまとめてつつ，高速化も計られています。

FBASIC

FutureBASICを使ったMasintoshでのプログラミング入門のサンプルです。PICTの表示を行います。

FLASH

MacromedeiaFLASHを用いたゲーム「ヘルメッ豚」です。元プログラムは古旗君，ビジュアル制作はYOUCHANです。

GALLERY

VisualLaboratoryのグラフィックデータです。今回は途中データも収録してみました。非常に大きなサイズのものもありますので，メモリの少ない方は多少注意してください。

JAVA

特集記事で作成したネットワークゲームのサンプルコードの一部です。稼働しているものとはおそらくバージョンが違います。最新版については直接サーバにアクセスして情報を得てください。

JAVASCRIPT

JavaScript講座のサンプルです。Macromedia FLASHで作成したデータを再生、制御していきます。

LOSTUNIV

株式会社ドーガが担当したテレビアニメ、ロストユニバースのCGパート映像と資料用画像です。MPEGおよび、JPEGファイルです。

MACSERVER

特集記事のMacintoshをWebサーバにしてCGIゲームで遊ぶためのサンプルプログラムです。Pearlで記述されています。

MESHV110

復刊号で付属したD3DMeshViewerの改良版ですが、今回の記事では使用されていません。1999初夏号(夏号?)で解説いたします。

QTVR

荻窪圭制作のQuickTimeVRのサンプルデータです。QuickTimeVR本体は含まれておりません。別途入手してください(www.apple.comなど)

QUAKE2

須藤氏制作のQuake2用Botです。実行にはQuake2が必要です。

SXBASIC

言語処理系を作ろう講座のサンプルです。X68000用のものとは違いますのでご注意ください。

TCLTK

Tcl/Tk入門のサンプルプログラム、ミニゲームなどです。一部のゲームはC言語で書かれたプログラムを呼び出していますので、そのままではWindows環境でしか動作しません。ご注意ください。

VB

VisualBasic入門のサンプルです。テキストエディタらしきものの例です。コントロールはセットアップしてから使用してください。

VC

VisualC++によるWindowsプログラミング講座のサンプルプログラムです。

XHANIM

HCPKPLAY.X(覚えてる人いるかなあ)の奈良原伸哉氏の新作投稿プログラムです。SATURNのCPKファイルとPlayStationのSTRファイルを再生します(それぞれ、複数の動画フォーマットが存在します。このツールはすべてのフォーマットに対する動作を保証するものではありません)。ちなみにLinux/X Window用です。一応オブジェクトも2種類(スレッド対応版とそうでないの)用意されていますが、そのまま動かない可能性もあります。ライブラリを揃えるなり、それぞれの環境にあわせてリコンパイルしてください。

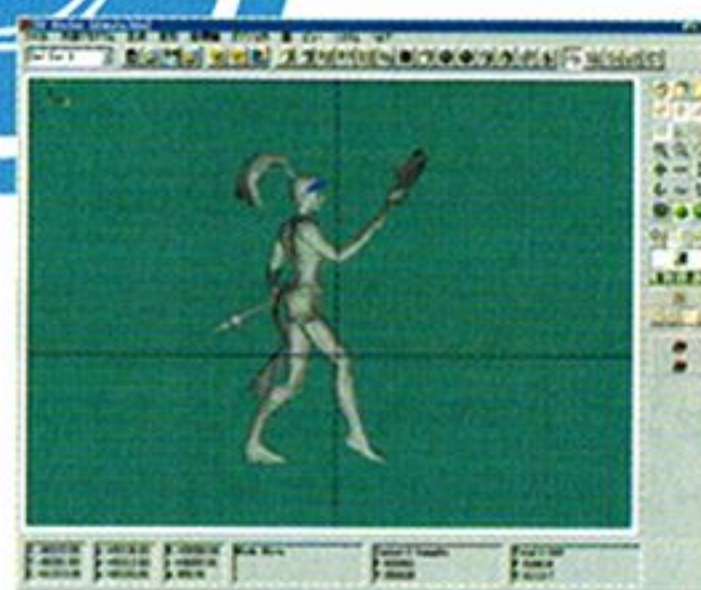
Black Disc2

Disk2

前回の付録CD-ROMでは記録内容に不良があり、一部の方にご迷惑をおかけいたしました。データ不良の内容につきましてはCD-ROM内のファイルをご覧ください。今回は前回のBlackDiscの内容のほとんどを収録いたしました。一部のツールはバージョンアップされております(EX68など)。

そのほかではマイクロネット3D Atelier ver.2.5体験版、Macromedia FLASH体験版。シマンテックVisualCafe ver.2.1体験版、およびver.3.0用アップデータが含まれています。

ATELIER

マイクロネット制作3D Atelier ver.2.5体験版です。格段に使いやすくなっています。

BLACK

前回のCD-ROM不良データ分の再収録です。

FLASH

ベクトルベースのオーサリング環境、Macromedia FLASHの体験版です。Macintosh用はSITファイルを展開して取り出してください。

VCAFE21W

シマンテックVisualCafe ver.2.1Windows用の体験版です。Java 1.x系の開発環境で、特集のネットワークゲーム開発で使われているバージョンです。

VISUALCAFEDE

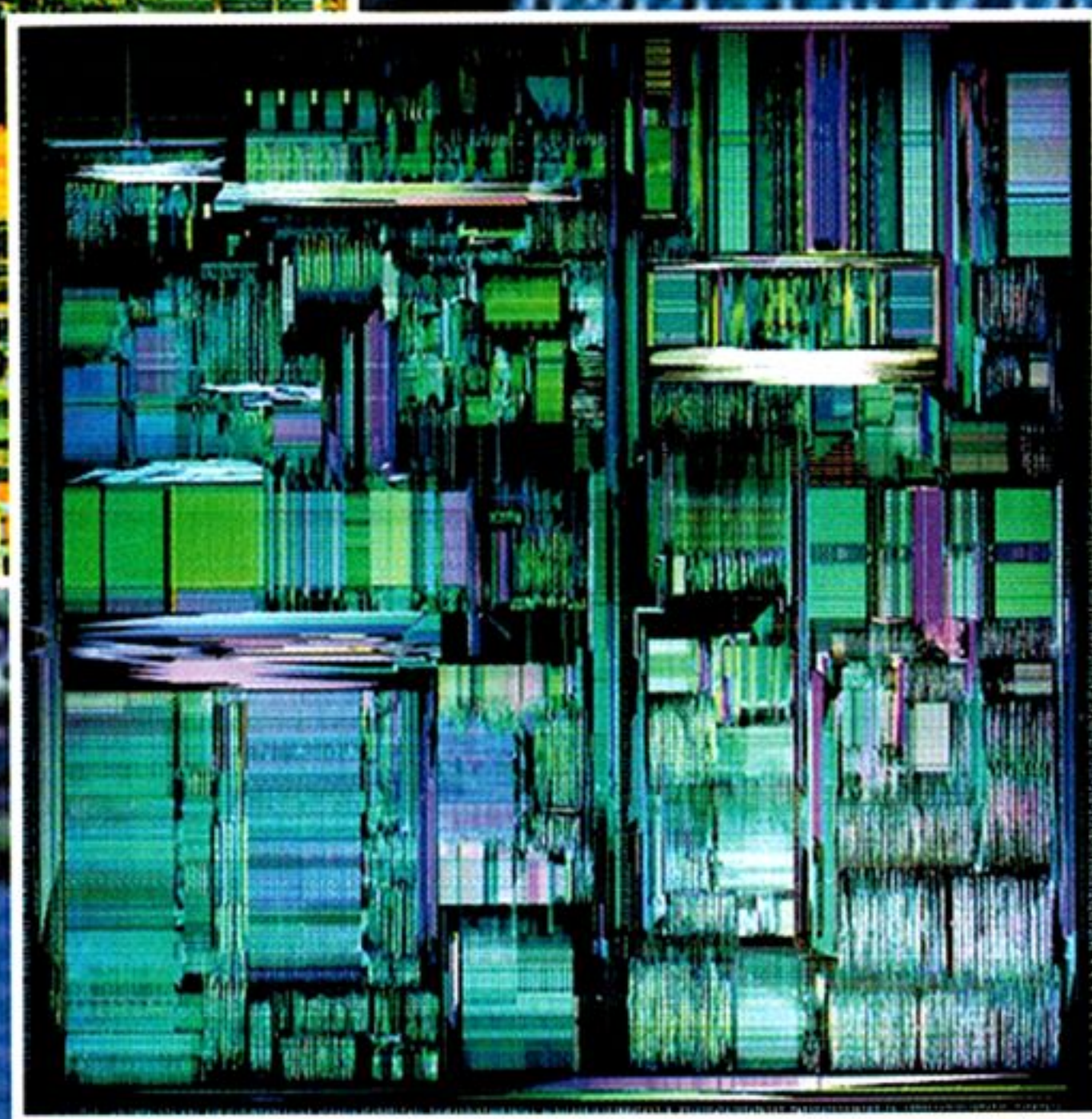
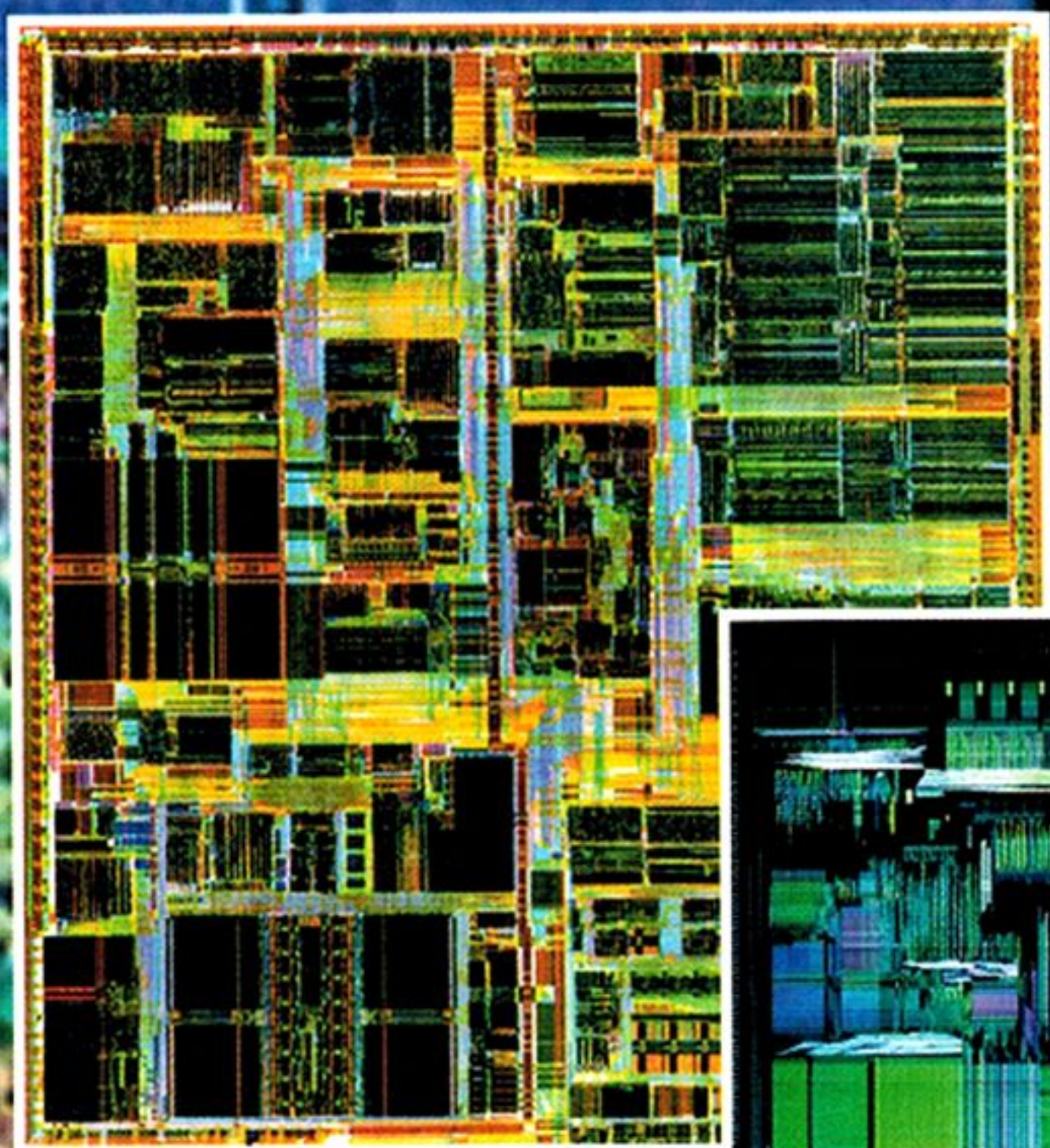
シマンテックVisualCafe ver.3.0 Database Edition用のアップデータです。VisualCafe ver.3.0が必要です。

VISUALCAFEPE

シマンテックVisualCafe ver.3.0 Professional Edition用のアップデータです。VisualCafe ver.3.0が必要です。

もう一度CPUについて考えてみよう

特別企画

CPU――なにをいまさらといった感じの人もいれば，どうやって動いているか見当すらつかない人もいるだろう。
世の多くの人にとっては，CPUの内部でどのようなことが起こっていて，どのような特徴があろうが大差はない。そういったことを理解していなくても，かまわない世の中になっている。PCなりゲーム機ではCPU部分がどういう風になっていようと，結局のところできることに大差はなく，性能差すらほとんど問題にはなってきていない。
しかし，より突っ込んだ話をしていこうとすると，CPUがどういうものであるかを理解していないとCPU以外のものがどういうものなのか，周辺機器がしている仕事はどんなものなのかすらよくわからなくなってしまう。DSPとの差異や3D表示での役割分担，ソフトウェアでできることとできないことなどなど……。
新しく初め直したおかげで，新旧の読者，それこそまったくの初心者から，OSが作れるくらいのレベルの人まで一緒くたに抱え込んでしまっている。当然，知識レベルもバラバラになってしまっている。
コンピュータの処理の基本的な知識，そして歴史的な背景などをまとめなおす必要がある。以下ではm RISC，CISCと呼ばれたチップたちにはどのような特徴があり，どのように変遷していったのか。現れては消えていくさまざまなCPUチップたちに思いをはせつつ，これから必要なのはどういったCPUなのかについて考えてみよう。

262 CISCとRISC，そしてメモリの物語 中野修一
266 VシリーズにみるCISCの幻想と伝説 中森章
274 VRシリーズの伝説MIPS RISCの系譜 中森章

特別企画 もう一度CPUについて考えてみよう

CISCとRISC そしてメモリの物語

中野修一/Nakano Shuichi

最近のパソコンでは，CPUを直接気にしてコードを書くことは少なくなっています。しかし，今後Oh!Xが目指していく方向にはCPU性能をフルに発揮させていくという道もあります。CPUの種類が問題なのか，使いやすさか，それとも種類など関係ない問題なのか……もう一度基本から見直してみましょう。

A「やっぱ，G3だよね」
B「あんた，コード書かないんだから関係ないでしょ？」

ま，多少きつい例だが，CPUを気にしてもしかたない人が妙にCPUについて口にする世の中になった。というよりは，CPUに関するいままではCPUの話題はシステム性能の話の一環でしか出てこなくなっている。G2だろうとG3だろうとPentium Ⅲだろうと K6-Ⅲだろうと，どうせ大差はない問題だ。

その一方で「九九式」のCPUが68060だということの反響が大きい(零式はCPU未定というのが公式見解かな？)。編集室に届いた愛読者ハガキなどを見てもいろいろな意見が載っている。もちろんほかのCPUがいいというものもある。

いちばん多く聞かれたのはPowerPCである(68060肯定派を除けばだが)。次いでSH4。Pentium系という人はほとんどいなかったように思う。皆無ではないが。傾向として，Power PCといってくる人は前述のような人が多く，Macのソフトが動くとか，68000のコードがエミュレートできるとか信じ込んでいる，まさかと思うような人もいるのに対して，SH4といってくる人は自分でコードを書くときのことも考えているように思われる。だいたいあの辺をガリガリやってみたいという感じ。うむ，気持ちはよくわかる。

そもそも，なんで68060のような入手しづらそうなチップなんだろうか？ モトローラ68Kシリーズの最高峰，CISCチップの到達点であるのは間違いないが，最近のチップほどの性能はどうやっても出ない。では，性能が必要なのだろうか？ その辺がいちばん焦点になる問題だ。九九式などは最初から68系のCPUを想定して企画されているものだ。当然，どの程度の性能が期待できるかも把握されている。であれば，CPUに負荷をかけないようなシステムを作るにはどうすればよいかという問題に注意が払われ，実質的にCPUはなんであってもあまり変わらないような構成が取られることになる。

「デモが必要だよね」
「ゲーム系の派手なのですかね」
「スプライトとかエフェクトバリバリの」
「でも，メインCPU全然動かないんじゃない？」
「うーむ」

こういったものだとサブCPUや周辺が担当する処理がほとんどになるので，メインCPUの出番はあまりない。OSやサブボードがどうなるかという点も未定なので負荷分担がどの程度になるかという肝心な点が不明なのだが，極力負荷を少なくするシステムにするのは間違いないだろう。

基本システムが開発しやすいということを第一に考えると，CISCプロセッサになる。

68000系なら開発者の数が全然違う。それを無視して，
「こっちのCPUのほうがいい」
というのは，
「ソースのコメントやマニュアルは全部ラテン語で書こう」
というのをいくら論理的に説得してもあまり効果がなさそうなのと似ている。いくらラテン語が美しい言語だと主張してもたぶん無駄だ。

68000のアセンブラは公用語なので，誰でも理解できる。BIOSのソースが上がればみんなでデバッグ可能だ。そういう特殊な環境だという前提を抜きに話をしても無駄だ。そもそもその前提なしには存在しえない話なのだから。

なんのことはない理由だが「わかりやすい」というCISCチップの最大の特徴を端的に示す例のようにも思われる。68000とはいえ実際にはそう明解でない部分もあるのだが，普及してしまえば勝ちだ。「アセンブラレベルで処理を書こうという気になれる」というのがCISCの最大のメリットだといってもいいだろう。

我々の多くはCISC CPUしか知らない(「iMac使ってます」とかいう人でCPUを知っている人はほとんどいない)。世はRISC全盛である。現状ではRISCとかCISCとかいう区別自体の意味もなくなっているのだが(PentiumやK6などの内部構造はRISCの方法論で高速化されている)，我々の多くは68000の10MHzないし16MHzしか知らない。いや，X68000ユーザーだった人ばかりではないのだろうが，386コードを書ける人などはほとんどいない(86コードは却下)。PC-98やTOWNSユーザーのごくごく一部でしかない。ATだとさらに少ないだろう。なぜなら開発環境自体が入手困難なのだ。なんでも揃っていて，情報もたくさんあるにも関わらず，一般ユーザーからは遠いところにしかない。

たとえ現在Pentium Ⅱ/400を使用していたとしてもCPUを意識して使うことはまずありえない。なにより，アプリケーションに至るまでには何重ものよくわからない機構が絡まっていて，プロセッサがどう動いているかなどは知るよしもない。

68000の10MHzとはどんなものだったかを思い起こしてみよう。まず，どんなに速い命令でも4クロックかかった。10クロックくらいの命令はざらだ。昨今のCPUだったら1クロック処理は当たり前で，下手をすると2，3命令が同時に処理される。クロックは30倍から50倍違う。こういう比較をすると，とんでもなく遅かったはずなのだが，さほど遅い印象を持っていた人はいないだろう。というか，昨今のシステムがなんでこんなに重いのかとかが不思議になることはないだろうか？

X68000に関しては，設計者がメモリウエイトを極力排除する方向で構成していたので，幸いにもメモリのウェイトを考えなくてもよかったことが挙げられる。X68030の段階でもページをまたがない限りノーウエイトだった。命令ごとの実行クロックはなんとか把握できる範囲であり，プログラマがほぼ完全に実行速度を把握することが可能だった。

RISC CPUは高度にパイプライン化されており，逆にいえば，パイプラインの停止は全体の実行速度に大きく影響する。

5段のパイプラインのプロセッサがあったとしよう。命令のスループットはすべて1クロックだ。実行ユニットの内部には5つの命令が1クロックごとにずらされた感じで同時に流れており，通常は1クロックで処理されているかのように動作する。外から見ると1命令1クロックしかかかっていないように見えるはずだ。20クロックあれば20命令分処理が終わっている。ここで，キャッシュが外れたとしよう。メモリアクセスサイクルが入るため最低1クロックプロセッサの実行が停止する。するとそこで動いていた5つの命令(各パイプラインに入っていたもの)はどれも2クロックで動作を終了したかのように見えることになる。全体としては5クロック分，損をしたことになる。

要するに，1ウエイトに対し，命令のスループットは100%×パイプライン段数分低下する，ということになる。68000だったらどうだろう？　1ウエイトに対し，スループットは最大25%低下する。もとから遅かった影響が少ないという話もあるが，こういう部分を完全にこなすからこそRISCは速いのだ。

メモリアクセスが発生しても，1次キャッシュに載っているデータはウエイトなしでアクセス可能だ(そういう風に作られている)。2次キャッシュ以降はメモリアクセスサイクルが発生し，プロセッサは停止してしまう。2次キャッシュに載っていなかったら，さらに長く停止することになる。これらのシステムではキャッシュがヒットしないという条件はあまり考慮されていないのだ。RISCはメモリアクセスを行うと負けである。キャッシュは当たって当然。RISCマシンであればキャッシュがちゃんと当たるようにシステムを作る必要がある。

データは使うたびにメモリからロードして，加工し，またメモリにストアしていく。という流れに一本化してプロセッサは設計されている。コンパイラはおそらく，レジスタを駆使し

Column　パイプライン処理とは

CPUなどのデジタル回路は，基本クロックに同期して一連の回路で処理した結果を次に渡して処理すると，たいてい2クロックかかる。最初のユニットでクロック，次のユニットで1クロックという感じだ(それ以上かかることももちろんありうるが，最近のデザインでは1ユニットで1クロック以上かかるようなものは世間が許してくれないだろう)。

CPUが処理する命令の実行というのはどういう工程になっているのだろうか？　まず，メモリ上にあるプログラムをCPUに読み込まなければ話にならない。これを命令フェッチという。メモリアドレスを指定して，そこにあるデータを持ってくる。メモリアクセスサイクルを考えなくてもCPUへの取り込みで1クロックはかかってしまう。

次に，取り込んだ命令がどんな内容かを判定しなくはならない。これを命令デコードという。命令には基本的にどの系統の処理か(オペコード)，どのレジスタを使うのか，どういうデータを使うのかといったこと(オペランド)が符号化されている。昨今ではひとつの命令はほぼ4バイト均一で扱われることが多い。32ビットプロセッサでデータを取ってくるときに都合がいいこと，可変長命令だと，命令フェッチなどが複雑になることなどの問題があるからだ。日立のSH2などは2バイト固定の命令長になっている。そのほか，2バイト4バイト混在のシステムも多いが，やや効率は落ちる(その分柔軟性があるともいえるが)。CISCマシンの場合は，「よく使う命令は短く」などという考え方もあったので(メモリが非常に高価な時期には有効だった)，命令のフェッチ，デコードは非常に複雑になっている。あとから追加された命令はどんどん長くなっていく。特にインテルプロセッサでは，32ビット拡張命令のためのプレフィクスコードなどを混在して含み，非常に複雑なロジックを必要とする。そこで高速化のために，命令をRISCコードやマイクロコードに分解するというプロセスを経ることで内部実行ユニットはRISCと同じ方法論が使えるようにしている。

それはともかく，取り込んだ命令は実行される。これも最低1クロックかかる。

そして演算結果を保存する。これも1クロックだ。

ということで，最短でも4クロックくらいというのが一般的なところになっている。ここで挙げたプロセスのそれぞれがパイプラインステージと呼ばれているものだ。それぞれは前のユニットでの処理結果を受け取って次に渡していく。バケツリレーのようなものだ。それぞれが1クロックで処理されるとしよう。命令フェッチが終わってデコードユニットに渡したときには，フェッチユニットは手空きになる。遊ばせておくのはもったいないので，次の命令を処理させてやる。それぞれのパイプラインステージでは，同時に別々の命令が処理されていることになる。出口では1クロックごとにそれぞれ結果が出ている。

このように1クロック単位で結果が出る場合，「スループットが1クロック」だといわれる。めいパイプラインの最初から最後までいくのに4クロックかかっていると(4段のパイプライン)，「レイテンシが1クロック」と呼ばれる。なにかの結果が出てくる時間と，特定の結果が出てくるまでの時間を表していると考えればいいだろう。

昨今のプロセッサでは，1つひとつのユニットを単純化していかないと，なかなか動作クロックを挙げられないという問題に直面している。実際には例で挙げたパイプライン構成よりも噛み砕かれたものになっていることが多い。

このようにパイプライン段数を上げたものをスーパーパイプラインと呼ぶこともあるが，ただのパイプラインと内容の差はない。

最近のチップでは8段パイプラインとか12段パイプラインとか，パイプライン処理とかかなり長いパイプライン構成になってきている。命令スループットは1クロックなので(最近ではほとんどが)，パイプライン段数が多くても別に高速であるわけではない。

パイプライン処理は複数の命令を同時に処理しているわけだが，命令の種類によっては，演算結果が直後の命令に影響するなどということはよくあることだ。最大のものは分岐命令である。

分岐命令などで，命令の流れが変わってしまうと，それまでパイプライン処理していたものが全部無駄になってしまうかもしれない。それ以前に，どこに処理が移動するかわからなければどの命令をパイプラインに入れていいのかわからないのでパイプライン処理自体が破綻してしまう。

昨今のプロセッサでは「パイプラインの流れを乱さないこと」が最優先されるので，分岐命令があっても実行される確率が高いほうをとりあえず実行しておくという方法を取る。これが投機実行(スペキュレイティブ実行)というもので，まあギャンブルみたいなものだ。

プロセッサの処理は2進法の世界なので，どんな場合でも分岐先は2つしかない。どっちかをやっておけば単純平均でもなにもやらないよりは得する確率が5割上がる。このギャンブルの勝ち率を上げるための処理が分岐予測だ。分岐処理のロジックがわかっている場合はコンパイラが予測の当たるようなコードを出すこともあるだろう。プログラムで非常によく使われる繰り返し処理を考えてみよう。1000回のループの場合，ループの末端の分岐命令は999回は戻り先への分岐，次の命令への分岐が発生することは1000回に1回しかないのだ。

パイプラインの実行とか，実行ユニットの構成とかは同じシリーズのCPUでも異なっていることがある。あるCPUに最適化されたプログラムだと，ほかのCPUでは実は効率が悪いかもしれない。「同じプログラムコードが動きます」というのが同系列のCPUファミリを採用する最大のメリットだが，なかなかそうもしていられないこともある。

そこでCPUに取り込んだ命令を並べてある順番では実行せず，結果に影響がない範囲でそのプロセッサに適したように並べ換えて実行するものもある。これはアウトオブオーダー実行などと呼ばれている。内部ユニットで処理の手空きのものがあれば，そこで処理してもかまわない命令を見繕って処理を回す(命令のディスパッチ)ことになる。

てこの流れを少し変えようとするだろうが，ハード設計自体は先ほどの流れをひたすら速くして性能を上げようというふうになっている。これをロードストアアーキテクチャという。データの流れの単純化により，プロセッサがそう動くならハード側もキャッシュで対応するなどシステム性能を上げる指針がつけやすくなる。

さらにただでさえ，RISCは命令数が多くなる。1命令1クロックということは，1クロックでできる範囲しか命令にしないということでもあり，ある処理は数クロック分に分けて実行されたりもする。とにかく，全体の流れがスムーズにいかないと非常に困るのは先ほど挙げたとおりだ。ひとつだけどうしても2クロックかかる処理があるなら，2命令を使ってそれを処理してやる。それぞれの命令は1クロック実行だけど，この2命令は必ずこの順番で並べて使用しないと使えないとかいった感じだ。一見，無意味なようだがパイプライン処理では効率がずいぶん変わってくる。しかし，命令取り込みのための単位時間のメモリアクセスも増えてくる。

そんなこんなをして，パイプラインを止めないように処理が進めらていくわけだ。そうして初めて高い性能が実現されることになる。

■G3 Macはなぜ速い？

1997年のISSC辺りでIBMからPowerPC750が発表されたときはひどくがっかりしたことを覚えている。当時，PowerPCマシンにはほとんど触れたことがなくても，そこらのMac系ライターの倍くらいはPowerPCを理解しているつもりではあった(なぜだろう)。ざっと見たCPUスペックはPowerPC604以上のものとは思われず，実行ユニットを減らし，パワーマネジメント機能を導入し，キャッシュコントローラを内蔵したという内容だ。

思えばPowerPCも不遇なプロセッサだったといえるだろう。名前どおりにパワフルなプロセッサなのにマシンに恵まれていない。

時を経て，PowerPC750はシリーズ名からG3(Generation 3)と呼ばれるようになり，Macintoshに搭載された。だいたいご存じの方も多いとは思うが，まさに絶賛という言葉がふさわしい大人気ぶりだ。

ずっと以前のOh!Xでも書いていたと思うが，PowerPCはスーパースカラプロセッサであるというのが最大の特徴だった。最初にPowerPC601，続いて廉価版のPowerPC603というプロセッサが開発された。601のほうが高性能とされていたが，実は内部ユニットの並列度では603のほうが整っていたのだ。違っていたのは……当時のソフトウェア技術者が601専用命令を使いこなしていたとはとうてい思われず，ひたすらキャッシュ部分の性能によっていたものと思われた。

いわゆるPowerMacintoshというマシンを思い出してみよう。システムコードの大半は68040のエミュレーションによって行われていた。どの道，速いはずはないマシンだった。

エミュレータの速度がひたすら問題になった。アップルご自慢のエミュレータは16KBキャッシュに最適化されていたので，キャッシュ容量の小さかったPowerPC603では有効に動作しない。

RISCではメモリアクセス速度が命だということはすでに述べたが，キャッシュ上のデータはノーウエイトで実行される。よって，キャッシュ内でどの程度のアクセスをまかなえるかはもっとも重大な問題だということになる。16KB用に最適化されているプログラムならば，8KBのキャッシュでは，絶対にもの凄く悪い効率でしか動作しない。場合によっては最適化されてないプログラムのほうがマシかもしれない。8KB用にエミュレータを組み直すべきだったのだ。アップルは603をリコールし，603eができあがった。603eについては悪い話は聞かないので性能上の問題はないのだろう。603と603eのキャッシュ以外の部分はまったく同じ構造だ。

PowerPC603は不当に低い評価を与えられていたと思う。IBMの心づもりとしてはおそらく603系列が主流になるとしていたはずだ。整数実行ユニットを3つも積んだPowerPC604は実際問題，あまり必要ないと思われていたのではないかと推測する。603eはそれなりに評価されているのかもしれないが，603でマイナスイメージがこびりついてしまっていた。

PowerPCを使ったマシンだと，ほかにBeBOXがある。BeBOXは2CPUシステムだった。しかしキャッシュの調停が面倒なんで最初からキャッシュを殺してあるという構成だ。

「でもPowerMacよりずっと軽快」

全然自慢になっとらんぞ。

「シングルプロセッサでいいから，試しにキャッシュを載せてみてくれないか？」

と開発者はいいたかったのではないかと推察する。BeOSはその後，シングルプロセッサのPowerMacに移植されたが，2CPUでないデメリットはあまりなかったとも聞く。マルチタスク性能は当然下がるのだが。

その後，初期のPowerMacではデータバスを32ビットしか使ってなかったということを知り，愕然とする。PowerPCは64ビットプロセッサのPowerから作られたチップであり，常識的に使えば外部バスは64ビットだ。32ビット長の基本命令を2個同時に取り込んでスーパースカラユニットに渡す。32ビットバスでは転送能力はそのまま半分になる。Pentium IIなどの最近のCISCプロセッサなら，1個の命令が内部で複数のRISC命令に置き換えられるので1命令ずつでも処理の並列度はあまり下がらないかもしれない。RISCだと細かな命令を連続的に与えるのは当然のこととしてプロセッサが設計されている。

1クロックで1命令あるいは1データしか取り込めない構成だと，プロセッサのパイプラインはスカスカになる。その辺のギャップはキャッシュによって埋められるのだが，バス幅が狭ければキャッシュリフィルにも時間がかかる(32ビットバス時のバースト転送って想定されてたっけ？)。PowerPC自体は，内部256ビットポートで32バイトのキャッシュ1ライン分を1クロックでフェッチできるという構成を持っていたり，キャッシュ周りの処理低下を最低限に防ぐような設計がちゃんとされているのだが……。ただでさえ遅いメモリアクセスがさらに遅くなったのではキャッシュの重要度だけがクローズアップされるのも道理だ。

PCI Mac(型番忘れた)からはバスが64ビット化されて性能も向上したらしいが，その時点でも劇的に高速になったという話は聞いていない。それは本来おかしなことだ。

そして，整数演算ユニット3本を持つPowerPC604という強力なプロセッサから，整数実行ユニット2個のPowerPC750を積んだマシンに世代交代が行われた。その結果は前述のとおりだ。よほどもともとのメモリ性能が低かったのだろう。すべてはCPUではなくてメモリ性能に依存していたわけだ。

■それでもアセンブラ？

RISC CPUはゲームマシンなどで一般的に使われるようになった。PlayStationの開発などでもメモリ性能の低さが問題になっているらしい。PlayStationはR3000(4KB命令キャッシュ)をCPUに積んだ構成だが，メインメモリへのアクセスが発生すると極端に性能が低下するため，レジスタ内での処理を完結させるコードを書くこと，スクラッチパッドRAMを活用していくことなどが性能アップの決め手となっているようだ。そうなるとアセンブラでコードを記述していかざるをえない(中森氏に聞くとメモリアクセスすると極端に遅くなるのはワークステーションでもなんでも同じらしいが。当然といえば当然か)。

RISC CPUとはいっても，構成はRISCと極端に大きな差があるわけではないのだが，内部処理をちゃんと把握しておかないと最適化が不

可能だというあたりがRISCの難しさになっている。単にCPUの構造だけ見れば，32本のレジスタ，対称性，特殊命令などの少なさなど，プログラムの組みにくそうな部分は少ないはずなのだが，オペランド2つを駆使するなど処理の考え方が細分されているので，やはりあまり人間向けのコードとはいえないだろう。まあ，これも単に慣れの問題という話はあるだろうが。

なんにせよ，OSなどのシステムはC言語などで記述されるからCPUはRISCでも問題ないだろうという意見

最低限の部分は誰かがアセンブラで記述しなければならない(BIOSあたりはCのコードよりアセンブラで書いてあるほうがいいなあ)。OSの上に乗っかったアプリケーションレベルの開発しかしないなら大した問題ではない。その下のレベルに対するアクセスを行うなら無視できないだろう。

では，ゲーム機などではないRISC環境で，どれくらいアセンブラは使われているのだろうか。

SPARC(無論ワークステーション)ではアセンブラを使わないとどうしようもないといっていた人もいるし(凄く大変そう)，Alphaではライブラリの書き直し作業などをアセンブラで進めている人たちもいるという。RISCになっても，こういう作業はやはり必要とされてくるのだ。

記述が68000系に近いとされるSH系では，比較的日常的(？)にアセンブラが使われているような雰囲気もある。限られたシステムで性能を出そうとするときには，やはり必須のものと考えていいだろう。

■マルチメディア命令

昨今のプロセッサの進化はマルチメディア命令の強化に向かっている。たいていの処理ではすでに現状のCPUで十分な性能を持っているといっていいのだが，特定の処理ではまだまだパワーが足りない。特にパワーを必要とするのはなにかというと「マルチメディア処理」とひとくくりでいわれているものたちだ。

それはグラフィックの加工であったり，音声処理であったり，動画再生補助であったり，3次元演算の強化などもその一環といえるだろう。

インテルのMMXなどはそのものだし(マルチメディアエクステンションの略ではないそうだが，マルチメディア用の拡張命令だ)，AMDの3D Now!，Pentium ⅢでのSSE，Power PCでもAltavecというMMXモドキが進行している。RISCワークステーションではもっと以前からマルチメディア命令が試用されていた。

こういったものが今後のプロセッサ性能を評価するときの決め手にもなってくる。ソフト側でサポートしていないとまったく意味がないので，一般ユーザーレベルだとアプリが対応するまで関係のない話なのだが，プログラムを作っていくという立場からは，使うと使わないとでは性能がまったく変わってくるので無視はできない部分である。

■再び零式へ向けて

X68000ユーザーは手でコードを書くという作業を行ってきた。実際にはコードを書くというより，コードを読む必要がある場合にCISCのほうが有利だったといえるだろう。零式ではすべてのソフトウェアはソース公開されていくだろうから，オブジェクトから解析を始めるということは少なくなるだろう。

世の中はRISC一色で，CISCは消え去るのは間違いない。また，極論すれば，どんなCPUでもできること自体はほとんど変わりはない。

一度作られたプログラムは，よくできたものほど，そのCPUに密着したものとなっているのでほかに持っていきにくい。が，不可能な話ではない。最近のCPU，特にRISC同士ならほぼコンバートだけでもいけそうな感じだし，レジスタ数の異なるCISCへもコンバートである程度まで対応できそうな感じはある。移植先のCPUのレジスタ数が多ければたいした問題はないのではないかと思われる。以前，電波新聞社が6809→68000，Z80→68000コンバータなどを使って移植効率を挙げていたことなどは有名な話だろう。C言語レベルでよければ，もちろんCPUはほとんど問われない。なにを選んでもかまわない。

おそらく最終的にはコスト，パフォーマンスそして趣味の問題となる。

九九式のブロック図を見ると載っているCPUは68060＋ColdFireという構成だ。ColdFireはモトローラの作った68040互換のRISCプロセッサ(RISCの定義などは一時的に忘れること)だが，68040に近い形態のものと，よりRISCっぽくモディファイされたものがある。将来性のあるのは互換性の薄いRISC寄りのタイプだ。

68060ディスコン以降はColdFireに移行という展開が予想される。実はColdFireならずっと高性能バージョンまでロードマップは展開されているのだ。多少RISCっぽく直すだけで性能は格段に上がってくる。そういう展開も含めて，CISCとRISCというものをもう一度見つめ直していくべきではないだろうか。

Column CPUとキャッシュについて

メモリアクセスが遅いとRISCチップは性能を発揮できないということは，RISCの産みの親であるIBMがいちばんよく知っていたはずのことなのだ。ROMPの事件はあまりにも有名だ。

世界で最初のRISCチップ801とその後に開発された商品版RISC ROMP。大きな違いを持つ両者なのだが，特に違うのはメモリアクセスだ。

801はECLプロセスで作成されたチップセットだった。RISCという考え方が非常に有効であることを示す高性能ぶりを発揮していた。それをVLSIに集積して改変したものがROMPという感じになる。非常に高価だったキャッシュメモリなどを載せられないので，ROMPはメモリと同じクロックで駆動されることになった。その結果，非常に遅くなってしまったのだ。

また，キャッシュということではWindows95発売時のフィーバーを思い出す。Pentium/133MHzマシンの頃の話だ。当時，インテルTritonチップセットの登場でPentiumマシンは軒並み高性能化していた。が，出せばなんでも売れる時期だったので，製品のばらつきも大きかった。CPUスペックしか見ない人の盲点だったのがキャッシュを搭載しているかどうかだ。2次キャッシュを積んでないメーカー製マシンなども平気で売られており，まともなマザーで組んだシステムと比較するとかわいそうなくらいの性能しか出ないということになった。もちろん，同クロックのCPUでだ。

最近の例で見ても，K6-2のベース100MHz化あたりからPentium Ⅱが高速だったのは2次キャッシュ部分がコアの半分の速度で動作していたからだというのがわかってきた。Celeron300Aの性能や，K6-Ⅲが現在x86プロセッサのほぼ最高峰にいることからも(Xeonを除けばか？)，2次キャッシュ部分＝メモリアクセス性能がシステム性能のほとんどを決めるという図式ははっきりしてきている。

ところで，最近のCPUでは2次キャッシュがコアクロックと同じで動作し，CPUと同じダイにまとめてられてきている。1次キャッシュと2次キャッシュでメモリとしての構造は別になにも変わらないという話も聞いた。1次キャッシュ上のデータはノーペナルティで命令実行サイクルに組み込まれるが2次キャッシュ上だとメモリアクセスサイクルが発生してしまう。

中森氏にうかがったところ，2次キャッシュは制御が単純でもいいけど，1次キャッシュなちゃんとやらないといけないということだったのだが，K6-Ⅲなどでは4ウェイセットアソシアティブを採用していたりするのだ。おそらくCPUコア部分の設計変更が必要になるからだと思われるが，特に根本的な障害はないのではないかということだった。製造工程での問題がなくなるとなると，今後は1次キャッシュを増やす方向に進んでくるのかもしれない。

特別企画 もう一度CPUについて考えてみよう

Vシリーズに見るCISCの幻想と伝説
日の丸MPUの系譜

中森 章/Nakamori Akira

かつてCPUを二分したRISCとCISCという2つの流れのなかで，CISC CPUの完成形といえるVシリーズを送り出したNEC。純国産32ビットCPUの誕生というドラスティックな事件とその行方を徹底調査でまとめてみました。その盛衰を通してCPUに求められているものを見つめ直してみましょう。

1999年である。巷では世紀末と騒いでいる人もいるが，世紀末は2000年だっちゅうーの。それはともかく，1990年代最後の年である。MPUが初めて誕生してからは四半世紀が過ぎ，日本でVシリーズなどのオリジナルMPUの誕生からも10年以上が過ぎている。

1980年代の中頃，日本のMPU設計者たちは1990年代の主役となるべくいろいろなアーキテクチャを考案した。それは，応用分野で必要とされる機能をすべて1チップに詰め込むという作業だった。まさにCISCの春だった。現状のインテルを初めとした米国独占のMPU市場への果敢な挑戦は歴史的な事件だったといっていいだろう。しかし，彼らの予想に反して，現在では単純な命令機能を持っただけのRISCが主流である。どこでなにを間違えたか。コンピュータの進歩は予測しがたい。

本稿では1980年代に提案され，歴史の必然の中で消え失せていった日本製MPUの興亡について，それがなぜ，どのように起こって，どうして失敗していったのか？　今回はその過程を徹底的に追いつつ，現在のMPUの潮流についてレポートしてみたい。これが，それらのMPUにとって鎮魂歌になれば幸いである。

インテルのi4004プロセッサのチップ写真。すべてはここから始まった

■日本で最初のMPU

MPUの歴史は，米国のインテル社4004の1971年の発売に始まったといわれている。これは，108kHzで動作する4ビットのMPUだった。4004は，四則演算を行う電卓程度の機能と速度しかなかったが，任意のプログラムにより，各種の演算や制御に使用することができ，電子回路の設計技術に革新的な変化をもたらした。

これがi4004だ

しかし，驚いたことに4004に遅れること数カ月，日本でもμPD700という2チップ構成のMPUが開発されている。これは，シャープがコカ・コーラ社から依頼を受けて論理設計をし，NECが製造した4ビットのプロセッサである。シャープは，最初，三菱電機に製造を依頼したのだが，これが見事にコケてしまう。そしてNECにお鉢が回ってきたわけだ。もし，シャープが初めからNECに依頼していれば，世界最初のMPUは日本製ということになっていたかもしれない。日本企業，特にシャープやNECはCPUの歴史の最初から表舞台のごく近くにいたことがわかる。自社使用の製品以外はほぼOEM生産だったシャープと異なり，以後，NECはMPUメーカーとしての挑戦を続けていくことになる。

μPD700は，その後NECが権利を買い取り，1チップのμCOM4として発売されている。このMPUは電子式キャッシュレジスタを中心に広く応用されたという。

4ビットMPUは，その後の低価格化の要求から，シングルチップマイコンが市場の中心になっていき，家電製品に採用されるようになる。この分野は日本の独壇場である。

■8ビットMPUの誕生

1972年，インテルは4004を改良し，8ビット化した8008というMPUを発表した。これ

は世界最初の8ビットMPUといわれている。動作速度は200kHzだった。そして2年後の1974年，インテルは8008にさらに改良を重ね，8080という8ビットMPUを発売した。これは，2MHzで動作し，0.64MIPSの性能だったという。

NECは8080の発表を受け，直ちにμCOM8という互換チップを製造した。NECにはμPD753というオリジナルMPUもあったが，国内ではこのMPUを本格的に利用できる装置はほとんどなく，自然と市場は海外に向かった。しかし，無名の極東のメーカーのオリジナルMPUを積極的に採用する顧客は皆無に等しく，セールスは苦労の連続だったという。そこで，NECは米国市場進出のため，セカンドソース路線も進めることとし，μPD8085などを商品化した。μCOM8の開発もその一環だったのだろう。ちなみに，インテルの8085は1976年に発表された。5MHzで動作し，性能は0.37MIPSだった。

■16ビットMPUの誕生

インテルが16ビットMPUの8086を発表したのは1978年のことだった。最初の版は4.77MHzで動作し，0.33MIPSの性能だった。その後，8MHz版(0.66MIPS)，10MHz版(0.75MIPS)が相次いで開発されることになる。8086のデータバス8ビット版である8088は1979年の発表である(IBM PCで採用されたチップだ)。

NECは16ビットMPUの世界ではμCOM16というオリジナルMPUをインテルに先行して商品化していた。1974年のことである。それに続き，1978年にはμCOM1600というMPUを商品化した。しかし，8ビットの経験から，当時90%以上の市場を持っていた米国をターゲットとするため，当面は8086/8088のセカンドソース路線を主力にすることにしていた。

■Vシリーズの登場

1980年代に入ると，日本のセットメーカーの技術力の進歩と優れた品質のおかげで，16ビットMPUを使用した日本製品が世界のリーダーとなることが期待された。市場を米国に求めていたNECもこの変化をあてにし，また，セカンドソースという，常に米国から一歩遅れた商品開発から脱却するため，1982年よりVシリーズの開発に着手した。Vシリーズの第1弾は8088/8086のクローンであるV20/V30(μPD70108/μPD70116)である。これらは1983年に発表された。8MHz(最大10MHz)で動作し，オリジナルである8088/8086の1.3倍の性能，というのが売りであった。

なお，VシリーズのVとはVictory(勝利)の頭文字である。当初はFuture(未来)の頭文字を取ってFシリーズになるはずであったが，Fが富士通を連想させるので没になったとか。

■V30の憂鬱

当時，8088/8086はNMOSのトランジスタ技術で製造されていた。NECはこれを，消費電力が少なく，集積度も上がるCMOS技術で製造することを検討していた。会社の方針がオリジナルMPUの開発ということになるにともない，そのCMOS化計画がV20/V30の開発に移行された。命令セットアーキテクチャは8086の次機種である80186(1982年発表)のものを基本的に採用し，ビットフィールドやパックドBCDなどのNECオリジナルな命令を追加した。さらに8ビットMPUからの移行をスムーズにするために8080のエミュレーションが可能になっている。

マイクロアーキテクチャ(内部構造)は大幅に変更された。8088/8086では1系統だった内部バスを2系統に変更し，マイクロコード制御を効率的に行えるようにした。また，HALT(STOP)命令実行時には電力の消費が1/10に減少する工夫もなされた。これが，オリジナルたる結縁なのだろう。ただし，内部バスの2系統化については，RISCの生みの親であるHenecyとPattersonの『コンピュータ・アーキテクチャ』という教科書の第1版の中でたいした効果はなかったと酷評されるのだが，それはのちの話。日経BP社から発行された日本語訳版では，その記述は意図的に削除されている。

閑話休題。1984年，NECはV20/V30がオリジナルであることを確認する裁判を米国で起こすが，これがインテルの逆襲にあってしまう。翌年には8086のマイクロコードを盗作したという容疑で逆提訴されてしまった。以後5年間にわたって裁判は続き，結局，マイクロコードの盗作の事実はなく，もし盗作していたとしても，著作権表示を怠ったインテルの負けという判決で結審した。

その後，NEC内部でも著作権の重要性を認識し，神経質なくらいにソフトウェアやマイクロコードに著作権(コピーライト)表示をするようになったのはいうまでもない。インテルとの裁判でV30の開発者であるK氏は米国では一躍有名になったが，その名前は日本のマスコミに登場することはなかった。もともとが8086のCMOS版として開発されたのがV30であるから，K氏がインテルのマイクロコードを見ていないということは，まずない。その証拠に，V30では，ドキュメント化されていない命令機能も完璧に盛り込まれていた。

V30に続き開発されたV25などではドキュメント化されていない未定義ともいえる命令はサポートしなかった。ところが，マイクロソフトのCコンパイラが，性能向上のためか，その従来は未定義だった命令を使用するようになり問題となる。V30ではそのようなトラブルはなかった。ところで，万が一，インテルとの裁判に負けた場合も考慮されていた。そのような事態に備えて，インテルのマイクロコードを見ていない人物がマイクロコードを書いた(いわゆる，クリーンルーム版の)V30も密かに開発されていたという噂だが，裁判が勝訴したので，その新V30は幻と終わったといわれている。

NECの大いなる野心を持って開発されたV30であるが，インテルとの裁判のおかげで，係争中はV30の使用を控え，様子見のメーカーが多かった。NECのパソコンであるPC-9801シリーズに採用された程度である。そして，裁判が終わる頃には16ビットMPUは，より高性能な80286の時代に移行していたのだった。80286は1982年に発表され，当初は6MHz，0.9MIPSの性能であったが，後に10MHz版(1.5MIPS)，12MHz版(2.66MIPS)が開発されている。

なお，Vシリーズの番号が20から始まっていることに疑問をもった人もいるだろう。実は，V10と呼ばれるMPUも存在した。Z80のCMOS版であるμPD70008がそれである。Z80をCMOS化しただけで取り立ててオリジナルな要素がなかったので正式なVシリーズとは認められなかったのだろう。

■V30の子供たち

V30は8080の命令コードを直接実行できるエミュレーションモードを持っていた。しかし，なぜ，と思った人も多いだろう。当時，8ビットMPUといえば，8080ではなく，8080の開発者がインテルを辞めて設立したザイログ社の開発したZ80が主流だった。当然，Z80のエミュレーション機能を持ったV30の要求も高かった。NECはμPD70008というZ80互換のMPUも製造しており，Z80のエミュレーション機能を持たせることは簡単なことだったはずだ。

はたして，そのようなMPUはNECホームエレクトロニクスのカスタムチップとして存在した。このMPUはPC88VAに搭載され日の目を見る(型番はμPD9002となっている)。PC88VAはPC8801シリーズとPC9801シリーズの両方のソフトが動作する可能性を秘めたパソコンだったが，政治的な理由で，一般的なMS-DOSとDOSコールのみを使用した(どうでもいいような)ソフトしか動作しなかったように記憶している(ちょっとあやふや)。

クリーンルーム版のマイクロコードを使用したV30とは別に，マイクロコードを使用せず，ワイヤードロジックで直接命令を実行するMPUも計画された。それが1988年に発表されたV33(μPD70136)である。16MHzで動作し，V30の約4倍の性能を有していた。ワイヤードロジックといってもPLAによるシーケンス制御であるが，PLAのパターンがマイクロコードのROMのパターンとどこが違うのかというのは疑問の残るところである。しかし，最初のうちは，PLAで動作するため，マイクロコードとは違って，細かい動作までV30と一致させることができなかった。そこで，パソコン用に，V30との互換性を高めたV33A(μPD70136A)が急拠開発された。V33AはPC-98DO+に採用され，従来機の2.7倍の性能を発揮した。またV33は，V30のあとを受けて，「ウルトラあっとプリンタ」以降，近年V830に取って代わられるまで，NECのワープロである『文豪』にも採用されていた。

■32ビットVシリーズ

インテル，モトローラなど先行する大メーカーのいる中で，NECがオリジナルのVシリーズに力を注いでいるのには理由があった。

ひとつは，1990年代の，日本の電子装置産業において，その商品競争力を強化するには独自のMPUが必要になること。もうひとつは，1990年代のプロセス技術の進歩と，それによって実現できる集積度を予測すれば，当時よりも洗練されたアーキテクチャのLSIを容易に製造できるようになり，1990年代以降でも変更せずに十分継承していける，新しいアーキテクチャを考える必要があったからである。

32ビットVシリーズであるV60/V70(μPD70616/μPD70632)のアーキテクチャは，32ビットシステム時代に焦点を合わせて設計されている。外部バスによって接続すると処理速度が低下するおそれのある，メモリ管理機能や，浮動小数点数の演算機能を，MMU，FPUという形で内蔵している。それによるチップの巨大化は，LSI製造技術の進歩によって初めて可能になった。また，大容量の情報を処理する要求から，小型で高性能な32ビットシステムが各分野に導入されていくにつれ，大量のデータを，安全かつ正確に処理することが重要になる。

V60/V70では，システムの信頼性を確保するために，FRMという誤り監視機能が組み込まれていた。また，V30で8ビットから16ビットへの移行を容易にするために8080エミュレーション機能が内蔵されたように，V60/V70ではV30のエミュレーション機能が内蔵されていた。ただし，V30の持っていた8080のエミュレーション機能はサポートされていない。このような革新的なアーキテクチャにもかかわらず，V30がインテルクローンであった経緯から，V60/V70も80386あたりのクローンであると思っている人が少なくなかった。V60/V70がいまひとつメジャーになれなかったのは，そんなところにも理由があるのかもしれない。

V60は1986年に発表された。その命令セットアーキテクチャは，過去のいろいろなコンピュータを参考に決定されたようである。命令セットの特徴は対称性と直交性に優れていることを第一とした。対称性とはすべてのデータ型に対して四則演算を初めとする演算が可能なこと，直交性とはすべての命令がメモリに対し任意のアドレッシングが指定可能なことを意味するらしい。この要求を満たすため，命令セットはVAX-11/780の命令セットを基本とした。それに加えて，V30などがサポートしていた文字列操作命令やビットフィールド命令，大型計算機であるACOSがサポートしていたビット列操作命令などが採用された。68000からはレジスタのプレデクリメント/ポストインクリメントアドレッシングが採用されている。マイクロアーキテクチャはV30を継承している。内部バスが2系統に分かれているところなどはV30そのものである。

さて，V60の瞬間最大性能は3.5MIPSである。16MHz動作であるから，1命令当たり4.6クロックで実行することになる。当時のミニコンであるVAX-11/780の性能が1MIPSといわれていた時代，3.5MIPSという性能はいかにも宣伝用という感が強い(性能がよすぎる)。NECはV60の平均性能は0.8MIPSであるとも発表している。

V70は，外部データバスが16ビットであったV60を32ビットバスにしたものだ。それと動作周波数が20MHzに上がった以外には，V60から大きな変更はない。当初の計画ではV70はキャッシュを内蔵した究極の32ビットMPUになる予定であった。しかし，1985年にインテルが発表した80386に対抗するため，V60からの改良をデータバスを広げることだけに絞り，急拠1987年に発表された。V70の瞬間最大性能は6.6MIPSである。これは，1命令を3クロックで実行する計算になる。V60/V70の命令デコードは3クロック周期で行われていたので，これ以上は逆立ちしても出せない性能だ。また，平均性能は4.3MIPSと発表された。80386の20MHz品の性能が4～5MIPSといわれていたから，とんとんの性能である。メジャーな80386に比べて，マイナーなV70が同程度の性能というのは，営業的にはちょっと苦しかったかもしれない。

■V60/V70の兄弟分

V60/V70の開発は，日本で最初の32ビットMPUの開発ということで，膨大なマンパワーが必要になると予想された。半導体事業部単独では開発が困難と判断したNECは，コンピュータ事業部のオフコン部隊に協力を依頼することになる。V60/V70はマイクロコードで動作するMPUであるから，マイクロコードを差し替えることで，オフコン用のMPUも同時に開発してしまおうという腹積もりだったようだ。このような機種展開はマイクロコード制御という特性を最大に利用したものといえる。はたして，V60/V70の兄弟分であるカスタムチップがいくつか誕生する。これらのMPUはNECのオフコンであるM10/M20に採用されたそうだ。

マイクロコードによる，別プロセッサの開発という観点からすると，エミュレーション機能も対象になった。噂によると，V60/V70はV30のエミュレーションだけでなく，80286や68000のエミュレーションも可能なハードウェア構成になっていたらしい。もし，エミュレーション機能がV30ではなく，68000だったら，V60/V70はもう少し注目されていたかもしれない。インテルのMPUよりもモトローラのMPUのほうがマニアや大学の研究者に人気があったのは確かなのだから。

■V80の悲劇

V80(μPD70832)はV60/V70のアーキテクチャを踏襲しつつより高性能なMPUを開発することを目標に始まった。命令セットアーキテクチャはもともと1990年代に通用する

ものを考慮していたのでV60/V70から変更する必要はなかった。

V80は，V60/V70と同じく，半導体事業部とコンピュータ事業部の共同で開発が開始され，その高性能の目標を2クロックマシンとすることに決定された。つまり，内部の各ユニットが，基本的に，すべて2クロックで処理を完了することが設計の指針とされた。また，2クロックで終了しない処理に対してもできるだけ実行時間が短くなるような工夫がされた。そのために導入された技術は，分岐予測機能とアドレス変換処理時のTLB入れ替えと文字列操作処理のハードウェア化である。

TLBとは仮想アドレスから物理アドレスへのアドレス変換時に参照される変換テーブルのことである。その変換テーブルにはすべての仮想アドレスが格納されているわけではなく，メモリ上のテーブルをキャッシングしたものになっている。アドレス変換時に，変換テーブルに登録されていない仮想アドレスが必要になると，メモリをアクセスして対応する物理アドレスをテーブルに格納しておかなければならない。この操作をTLB入れ替えと呼ぶ。

分岐予測は新規の機能であるが，TLB入れ替えと文字列操作処理は，V60/V70ではマイクロコードで実現していたものだ。また，命令に関しても，四則演算や論理演算のような基本命令はワイヤードロジックで実行させ，浮動小数点演算を高速化する機構や乗算器を内蔵し，高速化を図っている。さらに，NECのMPUとしては初めて，命令1Kバイト，データ1Kバイトのキャッシュを内蔵した。このキャッシュ容量は，現在のプロセス技術からすれば雀の涙であるが，当時としては大容量のほうだった。V60/V70がサポートしていたV30のエミュレーション機能はV80では削除された。システムの32ビット化が進み，16ビットからの移行を期待する必要がない，というのが公式な理由である。エミュレーション機能をサポートすると2種類のMPUを開発する労力が必要になることと，V80のリニアなアドレッシングとは相容れない，V30のセグメントによるアドレッシングのために余分なハードウェアを注ぎ込みたくないというのが本音であろう。

V80にとって悲劇だったのは，共同開発を行っていたコンピュータ事業部が，V80の将来性に不安を抱き，開発から撤退してしまったことである。また，ある事情からV80の開発責任者が開発から離れてしまったのも，その後の開発に混乱を与えた。しかし，そのような苦難を乗り越えて，V80は1989年に発表された。25MHz動作で，最大性能は12.5MIPSである。これは2クロックマシンということから導かれた(広告用)性能である。NECから発表された平均性能は9.9MIPSであった。その後，V80は33MHz品が開発された。33MHz品の最大性能は16.5MIPS，平均性能は13.1MIPSということであった。

1989年はインテルの80486が発表された年でもある。25MHz動作で20MIPSの性能は日本の半導体メーカーを慌てさせた。日本(NEC)では命令の2クロック動作を目指していたのに対し，80486は基本命令の1クロック実行をすでに実現していたからだ。

ともあれ，V80は某社の工業用パソコンに採用されることになる。80386からのリプレースだった。悲劇はその会社にV80を出荷したときから始まる。NEC内部の動作検証では特に問題がなかったV80であるが，実際の装置に組み込むとUNIXがまったく起動しないのである。そこで，複雑なUNIXの実行を追ってバグを突き止めるという作業が始まった。この作業は困難を極めたが，V80用UNIXの開発者やソフトウェアの専門家の協力でなんとかデバッグに成功した。しかし，最後までバグが残ったのは分岐予測機能である。NECは，しかたなくV80での分岐予測のサポートをあきらめることにした。分岐予測の性能向上への寄与は3%程度といわれていたので，それほど問題はなかった。また，某社でのデバッグの苦労からV80を一般に売ることはあきらめられ，ほとんど某社のカスタムチップとしての道を歩むことになる。

V80が市場に出回らなかったのは上述のような事情があったからである。開発者は品質(バグフリーということ)に自信を持っていたがNECの上層部が敬遠してしまったのだ。

■数値演算コプロセッサ

Vシリーズでは MPUだけではなく，周辺チップの開発も対象になっていた。V20/V30と同時期にはインテルの数値演算コプロセッサである8087に対応する上位コンパチのFPP(Floating Point Processor)の開発が始まっている。このFPPであるが，最初の仕様が壮大すぎたのか，開発を進めるたびにマイクロコードがチップ内のマイクロコードROMに収まらなくなり，機能を徐々に切り詰めて行かざるをえなくなった。そのたびに，製品名もμPD72091，μPD72191と出世魚(?)みたいに変更され，最終的には8087と同程度の機能しか持たなくなってしまった。

また，大幅な開発遅れ(V60が登場したときにもまだ開発中だった)のため，FPPは開発中止となってしまう。しかし，某社とはすでに商談がまとまっており，その対応は続けることになる。最終的にFPPは某社向けのカスタムチップとして生き残ることになった。某社のほかにもFPPを期待していたメーカはいくつかあった。それらに対してはV20/V30の隠し機能であった8087との接続機能を公開して対応したようだ。しかし，FPPの設計資産はV33用の浮動小数点コプロセッサであるμPD72291に受け継がれているようである。もしかしたら，V60AFPPのほうかもしれないが。

FPPのV60/V70/V80対応品がAFPP(Advanced FPP)である。このプロセッサもV60と同時期に開発が始まったが，開発は遅れに遅れ1988年やっと発表されることになる。現在，MPUにおける浮動小数点演算の基準となっているのはIEEE-754という規格である。FPPやAFPPはいち早くその規格を採り入れ実現を目指したが，開発遅れのため，NECで最初のIEEE規格の実現をV60の内蔵FPUに譲ってしまう。

V60/V70のマイクロコードROMには命令機能を実行するだけではなく，AFPPとのインタフェイスのためのマイクロコードが格納されている。しかし，このインタフェイス用のマイクロコードも巨大化してマイクロコードROMに入らなくなってしまった。そこで，V60/V70の本来の命令実行用のマイクロコードが削られることになった。犠牲にされたのはエミュレーション機能用のマイクロコードである。ビットフィールドやBCDストリングなどNECオリジナルの機能が削除されてしまった。はたして，V30エミュレーション機能は80186エミュレーション機能に成り下がってしまうのだが，NECオリジナルの機能は使用されることが稀れだったので，あまり問題にはならなかったようだ。

マイクロコードROMの容量が不足したのはV60だけである。V70には容量に余裕があったが，互換性という観点からV70のV30エミュレーション機能からも対応するマイクロコードが削られてしまう。マイクロコードの開発者はさぞ無念だったことだろう。ところで，V30エミュレーション機能を削除してもまだROM容量が不足していた。そこで，AFPP側の機能も削減されることになる。V60/V70の汎用レジスタとAFPP内のレジスタの間の転送命令が犠牲になった。AFPPをよく知る人(そんな人いないか)は，なぜAFPP内部のレジスタには値をメモリからし

か転送ができないのか不思議に思ったかもしれない。それは，上述のような理由による。

V80ではAFPPとのインタフェイス用に専用のバスが設けられる。これで，V80の浮動小数点演算が飛躍的に高速化されるはずだった。しかし，V80とAFPPのインタフェイスのオーバーヘッドから，四則演算に関してはV80の内蔵FPUのほうが高速であるという結果になる。V60/V70では内蔵のFPUの処理速度が遅かったのでAFPPにもそれなりの価値はあった。しかし，V80においてAFPPは行列演算や三角関数などの超越関数を実行するためだけに有効なプロセッサに成りはててしまう。数値演算コプロセッサとしては少し不様である。

■ V60/V70のその後

日本市場だけでなく世界のリーダーとなるべく開発された32ビットVシリーズであるが，EWSやパソコンという主要な応用分野ではインテルやモトローラなどの米国のメーカーを脅かすまでには至らなかった。いきおい，V60/V70はそれまで手垢のついてない組み込み制御分野やアミューズメントゲームの市場に活路を見出そうとした。その糸口として，V60/V70をシュリンクしてチップサイズと消費電力を削減した，安価なプラスチックパッケージ版が開発された。そして，この廉価版のV60/V70ではMMU機能とエミュレーション機能のサポートを行わないこととした。なにもその機能をMPUから削除したわけではない。LSIテスタによる，それらの機能のテストを省略することで人件費を減らし価格に反映させたのである。

まずは，ゲームメーカーのセガがV60の採用を決めた。その理由は，アーケードゲームは新製品を出しても，すぐに外国にコピーされてしまうので，広く知れ渡っていないMPUであることが採用の決め手になったという，NECにとっては少し複雑な思いのする理由だった。かくして，V60はセガのシステム基板であるSYSTEM32としてゲーム業界にデビューすることになる。その第1弾はRadMobileという自動車ゲームだった。セガの側でもゲーム機初の32ビットということで大々的に宣伝していたので覚えている人もあろう。しかし，第2弾がスパイダーマンだったのには拍子抜けさせられたが。

セガはV60に続いてV70もゲーム基板に採用した。これはModel-1という名だったか。SYSTEM32に比べて5倍の処理能力が売りであった。

セガがV60の採用を決めるや否や，カプコンがV70の採用を決めた。セガへの対抗意識で少しでも性能のよいMPUをということだったのだろう。カプコンからの要求は，最初コピーガード機能を備えたV70のカスタムチップということだったが，技術的な問題からカスタム化の話は消えたようだ。V70に関しては，そのほかにもセタやジャレコなどいくつかのゲームメーカーに採用されている。アーケードゲーム市場の旨味に気づいて，ほかの32ビットMPUが次々と参入してくる前の話である。

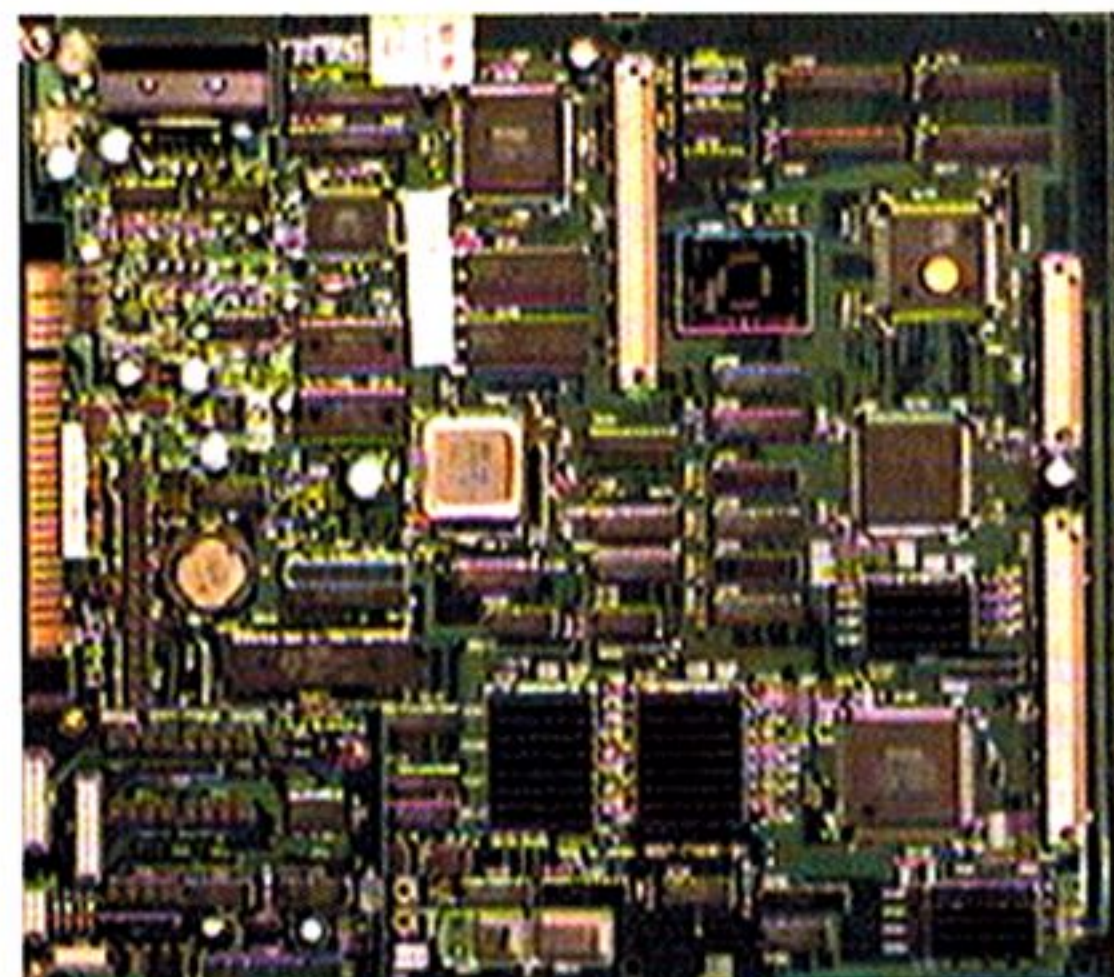
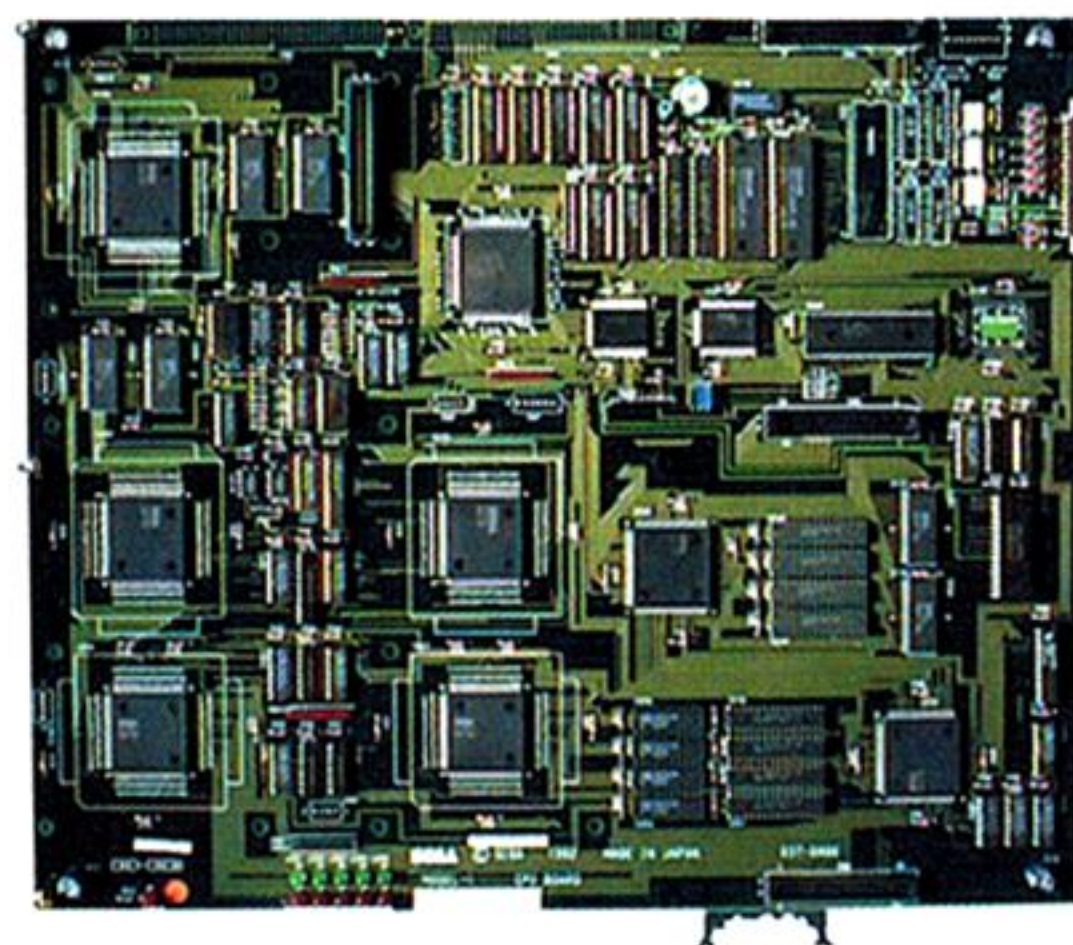
家庭用ゲーム機の世界にもV60は参入を試みた。ターゲットはNECホームエレクトロニクス（NHE）のPC-Engineの次機種だった。周知のとおり，PC-Engineとはゲームメーカーのハドソンが開発し，NHEが製造していたゲーム機である。現在では家庭用ゲーム機では常識のようになっているCD-ROMを早い段階で採用するなど，なにかと話題の多かったゲーム機である。しかし，PC-Engineの権利の大部分はハドソンのものであり，NHEとしては売り上げの割には少しの利益しかもたらさない商品であったという。そこで，NHEはPC-Engineの次機種は自社の主導で開発しようとしたのだ。

その頃，ハドソンはハドソンの側で独自にPC-Engineの次機種（PC-FXの前身）を考えていたのだが。ともかく，次期PC-Engineに関してはインテルからも売り込みがあった。インテルは当時開発したばかりの80960で勝負にきた。資料によれば，80960の発表は1988年で，20MHzで動作し，7～8MIPSの性能を持っていた。16MHz動作，3.5MIPSのV60では太刀打ちできないのは目に見えていた。しかし採用は公正に行われ，NHEが提出したベンチマークの性能によって比較検討されることになった。そして発表の日。NHEが選んだのはV60のほうだった。ベンチマークの性能は同等であったが，NECのほうが密な関係で開発を進められるという理由でV60が栄冠を獲得したのだ。一説には，非常に高度な政治的圧力がNHEにかけられたというが，真相は薮の中だ。

晴れて，次期PC-Engineに採用が決定したV60であるが，世の中の不景気が悪影響を及ぼした。NHEの事業が縮小化され，PC-Engineの自社開発の話が流れてしまったのだ。その後，またもハドソン主導によるPC-FXがPC-Engineの次機種として発表されるが，その末路はご存じのとおりである。

V60はNHEのワープロである『文豪』にも採用された。そのビット列操作命令がアウトラインフォントの展開に便利という理由だった。また，それまでの『文豪』のMPUにはV30が採用されていたので，V60のエミュレーション機能を使用すれば，従来のソフトウェアがそのまま流用できるという目論見もあったのだろう。V30の動作速度がまだ10MHzだった頃，V60のV30エミュレーションは13MHzのV30の性能に匹敵したといわれている。

実際にNHEで行われたV30エミュレーション機能のデモはおおむね好評だったという。NHEの素直な感想は，まさかここまでエミュレーション機能が動作しているとは思わなかったというものだったと聞いている。

しかし，NHEは安全を見込んでワープロ本体のMPUにはV33を採用した。そして，V60はアウトラインフォントを展開するという役割のためだけに採用されることになった。V33とV60の2個のMPUを搭載した『文豪』の誕生である（型番は失念した）。その売りは，それまでの「あっとプリンタ」を引き継ぐ，「ウルトラあっとプリンタ」だった。しかし，その後，V60には屈辱的な仕打ちが待っていた。性能評価の結果，なにもV60を使用しなくてもV33だけで，同等の性能が出せるというこ

SEGA SYSTEM32

SEGA Model1

とが判明したのだ。その後，V60は不採用になり，V33を1個使いした『文豪』が登場することになる。

V60をパソコンMPUに採用するとの話もあったようだ。それを計画したのはハドソンである。計画によるとシャープのパソコンであるX68000のMPUとしてV60を使用したパソコンを商品化するというものだった。この計画はなにがどう転んだのかは知らないが，のちにX68000用のV70ボードという形で実現することになる。V60のX68000（この場合XV60かな）というパソコンが登場していたかもしれないと思うとわくわくものだが，CISCは所詮消え行く運命であることを考えると，それはそれでよかったのかもしれない。

V70について聞いた，ちょっと毛色の変わった商談をひとつ。当時は，少し夢物語みたいな話だったが，宇宙開発事業団（NASDA）が2000年だか2001年に打ち上げるロケットにV70が搭載されるという話があった。といっても，ロケットを制御するわけではなく，宇宙ステーションの建設計画（JEMとかいった）があって，そこで使用されるEWSのMPUとして使いたいという話である。なぜ，V70が採用されたか。これは日本製のプロセッサであること。すでに枯れた（バグが出尽くして安定した）プロセッサであることが理由であった気がする。NASDAは国産の技術しか使用できないのでV70に白羽の矢が立ったわけだ。NECはその商談を受け，宇宙向け（つまり放射線などに強い）のV70を開発してNASDAに納めた。1990年代の前半のことだったか。去年，なにげなくテレビを見ていたら，NASDAの宇宙ステーションの特集をやっていた。しかし，そこで紹介された宇宙用MPUは，なんとTRONチップだった。V70はいずこに。

■TRONチップの台頭

TRONプロジェクトは1984年にスタートし，1990年代の半導体技術の進歩を前提として，MPU，OS，マンマシンインタフェイス，ネットワークに至るコンピュータの体系をすべて再構築しようというプロジェクトである。組み込み用OS（ITRON），パソコン用OS（BTRON），大型コンピュータ用OS（CTRON），分散処理におけるネットワーク管理のためのOS（MTRON）を作るためのプロジェクトと，これらのOSに最適なMPUを開発するTRONCHIPプロジェクトに分かれている。このTRONCHIP仕様に基づいたMPUがTRONチップである。

TRONチップのアーキテクチャは，1986年当時，従来のアーキテクチャ（すなわち，インテルやモトローラのアーキテクチャ）の寿命が尽きようとしているのを睨み，1990年代に通用し世界の標準アーキテクチャになることを目指している。TRONの提唱者である坂村健氏は，TRONチップのアーキテクチャは従来の32ビットMPUとはまったく別の新しいアーキテクチャであることを強調していたが，本質はV60のそれと変わらず，目新しいものではない。

実際に命令セットを眺めてみればわかるが，ほとんどがV60からのパクリである（まあ，そもそもV60がVAXのパクリという説もあるが）。その特徴は，対称な命令体系と，多段間接アドレッシング，異種データ型間の計算機能である。異種データ間の演算もなんだが，多段間接アドレッシングにより，パイプライン処理を妨げる原因のひとつであるメモリ間接アドレッシングが（理論上）無限回のメモリ参照まで許すように拡張されていたのには笑えた。ITRONなどのOSは現在でも使われているところを見ると，坂村氏はOSのアーキテクトとしては一流かもしれないが，MPUのアーキテクトとしてはセンスにピントはずれなところがあったのではないだろうか。

それはともかく，最初のTRONチップは日立によって製造されたGMICRO/200で発表は1988年である。1Kバイトの命令キャッシュ，128バイトのスタックキャッシュ，16バイトの分岐予測テーブルを内蔵し，20MHz動作で6MIPSの性能を出した。当時，日立はモトローラの68000のセカンドソースをしていたが，GMICRO/200の開発にはそのハードウェア技術が流用されたと思われる。TRON CHIP仕様では最初MPUはリトルエンディアンだったが，実際に出てきたプロセッサはビッグエンディアンだった。これは，68000の内部をよく知る日立のエンジニアの意見が仕様に反映されたと見てよいだろう。

1989年には富士通からGMICRO/300が発表された。COBOLでの使用を想定し，10進演算を効率よくサポートしているのが特徴である。性能は33MHz動作で4.5MIPSという。同じ年，三菱電機からはGMICRO/100が発表されている。これは，キャッシュとMMUを削除し，分岐予測を採用した組み込み制御用途のMPUである。25MHzで4.5MIPSの性能である。GMICRO/100はTRONチップのなかでいちばん出来がよかった。性能も高いし，なによりもレイアウトが非常に美しかった。三菱電機では，それを誇りに思いGMICRO/100のチップ写真をジグソーパズルにして，社員に配ったくらいである。

日立，富士通，三菱電機はGMICROグループを結成し，TRONチップをお互いにセカンドソースしあっていたが，この3社のほかにもTRONチップを製造していた会社がある。松下，東芝，沖電気である。日本のLSI製造メーカーが一丸となってTRONチップにより，NECのVシリーズに対抗しようという構図が見え見えである。

特に，沖電気が発表したO32というTRONチップは，命令キャッシュ1Kバイト，データキャッシュ1Kバイト，分岐予測機構，高信頼化システム対応と，意欲的なアーキテクチャだった。性能は33MHzで10MIPSを目標としていた。しかし，これはNECのV80の機能をそのままなぞったようにしか思えない。マイコンショーでO32を見たとき，V80の亡霊を見たような気がした。そして，デバッグが大変そうだなと感じたのを思い出す。事実，O32が世の中に出たという話はいまだ聞いていない。

TRONチップの最後（？）を飾ったのが，富士通から1993年に発表されたGMICRO/500である。スーパースカラ技術を採用し50MHz動作で100MIPSを目標としていたが，このMPUの消息も知らない。

TRONチップの全盛期は1989年までである。1990年代のMPUを目指して開発されたにもかかわらず，1990年に入ると各メーカーは次々とTRONチップから撤退していった。そして，メーカーはRISCチップのセカンドソースに力を注ぐことになる。RISCはEWSなどの限られた分野にしか採用されない，という坂村氏の主張とは裏腹に，世界のMPUはRISC一色になっていく。いまとなっては「1990年の終わりにはTRONチップが世界の主流になっている」という，日経エレクトロニクス誌上での坂村氏のインタビュー記事での発言が虚しく響くだけだ。

■V80からV800シリーズへ

V80の開発を最後にNECのVシリーズは終わりを告げる。しかし，V80の火は消えてはいなかった。ターゲット市場を組み込み制御分野に絞り込み，フットワークを軽くしたV80の改造品が計画された。具体的には命令の直交性を犠牲にし，効率よく命令実行ができるようにした。つまり，メモリとメモリの間の演算やメモリ間接アドレッシングを禁止して，レジスタとレジスタ，レジスタとメモ

リの間の演算のみを許すようにしたのだ。V80の命令体系の特徴である直交性を捨て去る決断をさせたのは，当時，RISCチップが市民権を得てきていた事実による。

また，削除した命令形式やアドレッシングはCコンパイラで使用されることは稀だった。メモリとメモリの間の演算やメモリ間接アドレッシングは，本来はCコンパイラの設計を簡単にするために導入されたものだが，それらの機能を使用するとアプリケーションプログラムの実行速度が低下するというジレンマがあったからだ。

それはさておき，このローエンド向けのMPUはV80Lと呼ばれていた。しかし，所詮はCISCの改良版であるV80LはRISCと比べると性能の限界が見えていた。市場に与えるインパクトも少ないと判断され，V80Lの開発は凍結されることとなる。その代わり，V80からアーキテクチャを継承した，オリジナルなRISC MPUを開発しようという動きが起こる。これがV810（μPD70732）の開発の契機になる。

当時は米国でHenecyとPattersonの『コンピュータアーキテクチャ』というMPU開発の教科書が発行されたばかりであった。この本により，多くの技術者はそこで述べられているコンピュータ設計における定量的アプローチの大切さを再認識させられる。それまで直観だけでアーキテクチャが決定されていたことがいかに多かったことか。そして，その教科書で説明されていたDLXという仮想MPUがMIPS社のR2000/R3000のモデルになっている。また，この教科書を読むと32ビットRISCのアーキテクチャはR2000の範疇から逃れられないような感覚を持ってしまうのも事実だ。つまり，R2000のアーキテクチャと大きく異なるMPUは作れなくなってしまうのである。特に命令長を32ビット固定とすると，自然とR2000と似通ったものになってしまう。

この呪縛から逃れるため，NECでは命令長が16ビットの32ビットRISCの開発の検討を始めた。組み込み制御分野では，アプリケーションプログラムはROMに格納されるため，プログラムサイズは小さいほうがよいからだ。ただし，コンパイラが最適な性能を発揮するにはレジスタの本数は従来の32本を維持したい。レジスタとレジスタ間の演算の場合，ソースオペランドとデスティネーションオペランドの指定が必要なので，そこで，5ビットずつ計10ビットを消費することになる。残りのビット数は6ビット。最大64種類の命令を指定することが可能だ。ということで，16ビット命令長のメドはついていた。ただし，ロード/ストアの場合はレジスタで指定するベースアドレスに対するオフセットとして16ビットは必要となる。遠くへの分岐に関しても16ビット長だけでは指定できない。などの検討の結果，新規アーキテクチャは，基本の命令長を16ビットとしながらも32ビット長の混在を許すという方針で決着した。

この方針の下，32ビットRISCであるV810の開発が開始された。RISCであるから，命令の処理は当然1クロックである。これらはハードワイヤードロジックによって実行される。ただし，浮動小数点演算（これは，組み込み制御分野では倍精度は必要なしと判断され単精度のみのサポートとなった）とビット列操作命令（いつの間にかこれがVシリーズの最大の特徴となっていた）はマイクロコードにより数クロックかけて実行される。

さて，V80からの継承性はどうなったのか。これは，アセンブラの記述において命令のニーモニックを極力あわせられるようにした。いわば精神的な継承である。これにより，V60以来培われてきたVシリーズのコンパイラ技術を容易に流用することができる。

かくして，32ビットRISCはVシリーズに続くV800シリーズと命名され，1992年にその第1弾としてV810が発表されたのである。V810は25MHzで動作し，18MIPSの性能だった。V810のそのほかの特徴としては，16×16ビットの乗算を1クロックで実行するDSP機能，ローパワーモードなどが挙げられる。翌年には，コストパフォーマンスを追求した，外部バス16ビット版のV805も開発されている。思えば，V810も悲劇のMPUである。任天堂のスーパーファミコンのCD-ROMドライブの制御用に決定していたのに，CD-ROMドライブ企画自体が没になった。同じ任天堂の立体ゲーム機であるバーチャルボーイに早い次期から採用が決まり極秘裡に開発が進められていたが，蓋を開けてみると大コケした。PC-Engineの次機種であるPC-FXにも採用されるが，これまた大コケしてしまう。どれも，営業が他社との競合の中で苦労して勝ち取ってきた商談だけに，営業サイドの落胆の様子は推して知るべきだろう。

NECがV810のセールスに必死になっている頃，日立でも16ビットの命令長を持つ32ビットRISCチップを開発していた。これが，SH（SuperHitachi）である。SHシリーズの最初は1992年に発表されたSH-1である。性能は20MHzで26MIPSである。全命令16ビット固定長，DSP機能内蔵，ローパワーモードと，V810と変わらない仕様を持っている。応用分野が組み込み制御，形態端末などというのもV810と競合している。

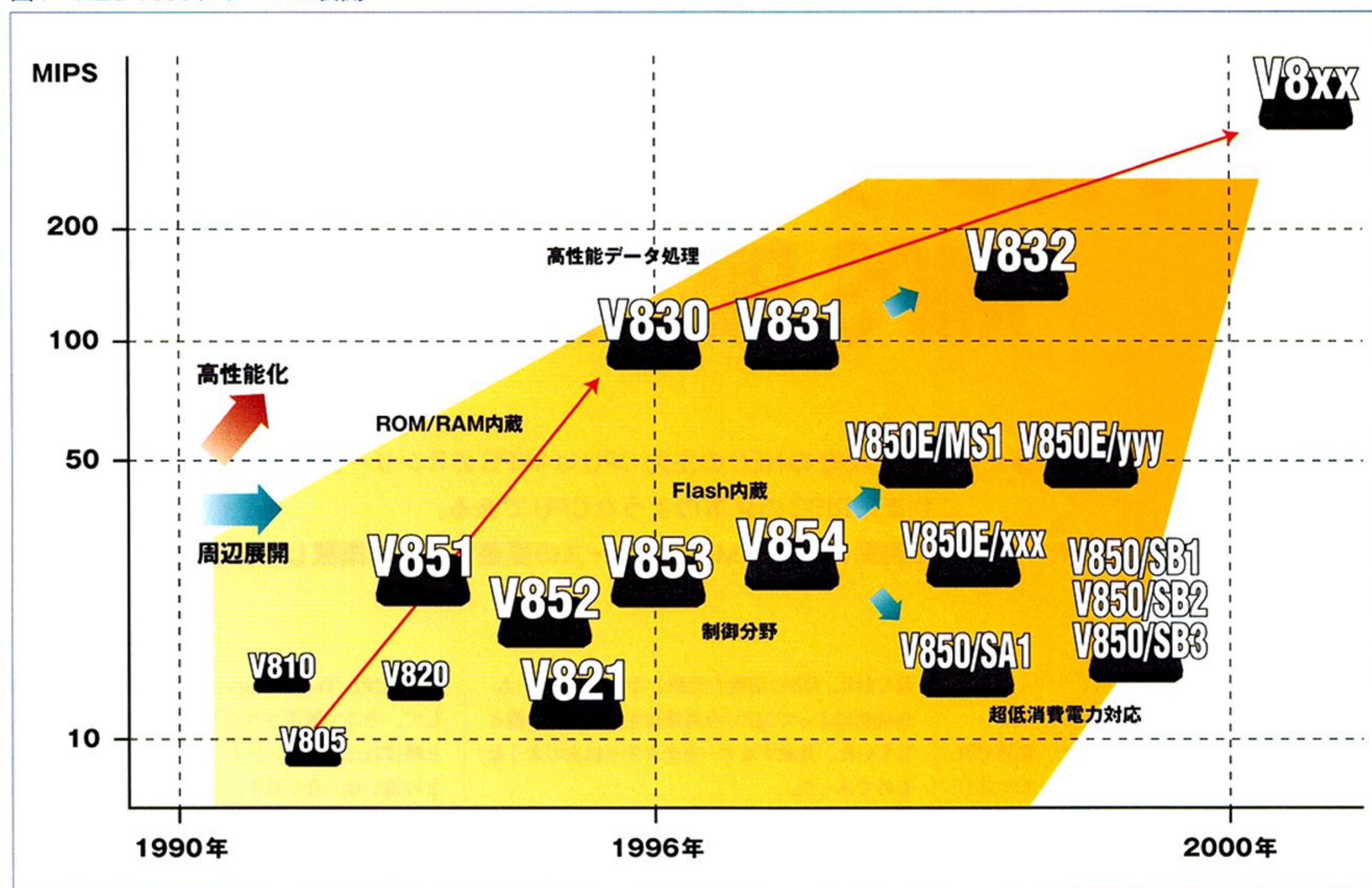
かくして，NECと日立は限られた応用分野での顧客獲得に向けて熾烈なシェア争いを始めることになる。SH-1に続き，日立はSH-2（26MIPS，20MHz），SH-3（78MIPS，60MHz）と，SH-1の改良品を次々と市場に投入していく。SH-1とSH-2はセガの家庭用ゲーム機であるSaturnに，最新のSH-4（360MIPS，200MHz）は同じくセガのDreamcastに搭載されているのでご存じのことと思う。

■ V800シリーズのその後

あるとき，それまでNECのシングルチップマイコンである78Kシリーズを使用していたハードディスクメーカーの某社が，78Kシリーズの採用を打ち切るという噂が流れてきた。某社は78Kシリーズにとって最大の顧客だったので，NECの上層部は大慌てだった。採用打ち切りの理由が，78Kシリーズの将来性への不安と性能の低さだったので，それを払拭すべくV810の登板となる。ただし，V810をそのままシングルチップの代用とするためにはアーキテクチャに不満があった。ひとつは，ロード/ストア命令が32ビット長なので，プログラムサイズが大きくなること。もうひとつは制御分野のプログラムでは頻繁に出現する単一ビットを操作する命令を備えていないことだ。そこで，16ビット長のロード/ストア命令と単一ビットの操作命令を備えたV810の設計が開始され，19xx年にV851（V850ファミリの第1弾，μPD703000）として発表された。性能は33MHz動作時に38MIPSである。V851はアセンブリ言語レベルではV810との互換性はあるものの，オブジェクトレベルでの互換性はない。16ビット長の命令を基本にしてはいるが，制御分野での命令の出現頻度を考慮し，命令のコード割り当てを見直したためだ。V850ファミリは，その後，V852，V853（フラッシュメモリ内蔵），V854（低電圧化），V85E0E/MS1（52MIPS），V850/SA1（超低消費電力化），V850/SBx（低ノイズ化）とラインアップを広げていく。

そして現在では，某社のハードディスクだけでなく，自動車のエンジン制御など，従来はシングルチップマイコンが使われていた分野に幅広く採用されている（みたいだ）。

V810にはもうひとつの流れがある。V810とオブジェクトレベルでの互換性を保ちつつ命令機能を拡張して，より高性能なMPUを

図1　NEC V800シリーズの展開

実現しようとする流れである。この流れの中で誕生したのが1996年に発表されたV830（μPD705100）である。

V830は応用分野を，組み込み制御の分野でも，特に，カーナビ，インターネット関連機器，マルチメディア端末，デジタルカメラ，カラーFAXなどに焦点を絞り，マルチメディア処理系の命令をサポートしているのが特徴である。ソフトウェア（ミドルウェア）だけで，専用LSIが不要なマルチメディアシステムの構築を可能とすることを目標としている。V830は100MHzで動作し，118MIPSの性能である。後継機種としてV831（周辺機能内蔵），V832（170MIPS，143MHz）などが開発されている。V830はV850とは異なり，必要に迫られて開発されたMPUではない。V800シリーズのなかでV810に続く次期MPUとして商品計画されたものだ。追加されたマルチメディア処理のための命令群は趣味的なものが多く，実際に活用されているのかは不明である。V830こそ，HenecyやPattersonが提唱した定量的アプローチに帰って命令セットを決定すべきではなかったのか。V830は数年前のマイコンショーで大々的にプロモーションされたにもかかわらず，NHEの『文豪』にV830，某社のカーナビにV831，某社のプリンタにV832が採用されている程度である。ほかにも細々とした商談はあるらしいが，ほとんど1社に1製品というカスタムチップの様相で，あまりパッとしない。

V830にMMUを内蔵して新たなステップへの移行も検討されたが，組み込み制御分野への高性能RISCチップの参入（NECでもVRシリーズが組み込み分野へ積極的なアプローチを続けている）に押されて，V830はその寿命を終えようとしているように見える。

■最後に

世界のリーダーとなるべく鼻息荒く打ち上げられたオリジナルのVシリーズとV800シリーズであるが，やはり世界の壁は厚かった。シングルチップコンピュータの分野でV850ファミリが健闘しているが，それ以外の分野では，世界標準のRISCチップにことごとく敗北していっている。読みが甘かったといってしまえば身も蓋もないが，ほんの一時期の間でも，他人が考えたのではない，オリジナルなMPUを開発していたことは，その開発者にとっては技術者冥利に尽きるのではないだろうか。米国のひとり勝ちという状況が続いているMPUの世界で，日本でもオリジナルなMPUが開発されていたことを，心の片隅にでも留めておいてほしいのだ。それにしても，Vシリーズの衰退を思えば，日立のSHって結構頑張っているなあ。

《参考文献》

1）日経BP社，『日経データプロ マイクロプロセッサ』，1994年廃刊
2）http://www.cs.uregina.ca/~bayko/cpu.html
3）http://www.super-h.com/tech/technical/fam1_1.html
4）元NEC勤務の某氏の昔話

（この記事はノンフィクションですが，伝聞などによる思い込みや誤解があることは否定できません。よって，いかなるお問い合わせにも答えかねますので，ご了承ください）

特別企画 もう一度CPUについて考えてみよう

VRシリーズの伝説 MIPS RISCの系譜

中森 章/Nakamori Akira

国産路線から離れた現在のNECの主力CPUはMIPSのRシリーズだ。
これまたRISCの見本のようなCPUである。
ここではプロセッサの製品展開についてMIPSシリーズの変遷とともに概観してみよう。

■はじめに

インテルベースのプロセッサに次いで，世界で数多く使用されているのがMIPS RシリーズのCPUである。ピンとこない人もいるかもしれないが，RシリーズはNintendo64やPlayStationに使われている石の種類だといったほうがわかりやすいかもしれない。一時はSATURNに3つずつ載っていた日立SHシリーズがRISCチップのトップだったのだが，さすがに勢いは少し衰えてきているようだ。

MIPSアーキテクチャのRISC MPUを製造しているメーカーはRISCチップのメーカーの中ではもっとも多いのではなかろうか。日本ではNECと東芝が競いながらMIPS RISCの提供を続けている。双方とも世界的に見ても非常に優秀なチップサプライヤーであり，最先端のチップを製造している。今後のキーとなるプロセッサについて豊富な関連情報が集まるのは非常に幸運なことといえるだろう。本稿ではNECが製造するMIPS RISCであるVRシリーズを中心にMIPS Rシリーズの概要を述べてみたい。

なお，この原稿では，NEC以外のMIPS系RISCは簡単にRxxxxと記述している。各社いろいろな名称がつけられていてややこしいからである。ただし，NEC製品に関してはVRxxxxとしている。

■MIPS社の現在まで

1980年頃，米スタンフォード大学においてRISCの研究がなされていた。米カリフォルニア大学のバークレー大学分校でもRISCの研究が行われており，RISC活動の先駆となった。どちらも，単純化によってMPUを高速化することを目標としていた。共通するアーキテクチャは次のようなものであった。

- ロード/ストアアーキテクチャ
- パイプラインを用い命令を1サイクルで実行する
- 遅延分岐

これは，現在のRISCチップに見られる特徴でもある。さて，スタンフォード大学のRISCはMIPS(Microprocessor without Interlocked Pipe line Stage)と呼ばれていた。その名称のとおり，パイプラインの動作をインタロックなどで停止させないことが特徴である。このとき，レジスタにロードした値は直後の命令で参照できない(遅延ロード)，分岐するときにパイプラインに1命令分の空きができるので，その時間に分岐の次の命令も実行してしまう(遅延分岐)などの制約がソフトウェアに課せられる。

ハードウェアはインタロックを制御せず，調整をすべてソフトウェアに任せるという考え方だ。制約が多い分だけ，ソフトウェアの最適化などは大変になる。MIPSではたとえソフトが複雑になっても，ハードをできるだけ単純にすることを第一とした。これは結果的に，コンパイラ技術の劇的な進歩を呼んだ。まがりなりにも，RISCというMPUが使いものになることが世間に認められたのは，コンパイラ技術の向上によるところが大きい。

1984年，スタンフォード大学のMIPS RISCを発売する目的で，MIPS Computer Systems(現在はMIPS Technology Inc.)が設立された。そして，そこで製造されたMPUはR2000/R3000と呼ばれた。スタンフォード大学のMIPS RISCとの違いは，命令長を32ビット固定としたこと，レジスタを32本にしたこと，例外処理をサポートしたことなどで，基本構成にたいした違いはない。R2000とR3000の違いは外部キャッシュの容量を128Kバイトまでサポートするか265Kバイトまでサポートするかである。R2000が実際に発表されたのは1986年のことである。8MHz動作で5MIPSの性能だった。1988年には16.7MHz品も発表されている。

ところで，MIPS社は，論理設計，回路設計などの開発とライセンス管理などを行うだけで，実際のMPUは製造していない。半導体製造メーカーにライセンスを供与し，MPUを製造してもらい，ロイヤリティで経営を成り立たせている。MIPSアーキテクチャのMPUの製造ライセンスを受けた半導体メーカーにはIDT，LSI LOGIC，Performance，Siemense，NEC，NKK，東芝，ソニーなどがある(最近，IBMなど新たに3社が加わった)。

MIPS社が最初にライセンス供与を始めたR3000は，これらのほとんどのメーカーが製造していた。しかし，R4000からはメーカー間に技術の差が出始めた。MIPS社が想定した微細な製造プロセスを使用できないメーカーが出てきたのだ。結局，R4000はIDT，NEC，東芝のみが製造することになった。これ以来，MIPS社は各MPUごとの製造プロセスにあうメーカーに製造をさせるという方針に切り替えた模様である。たとえば，R4200はNECのみ，R4600はIDT/東芝/NKKのみ，R5000はIDT/NECのみ，R8000は東芝のみ，

R10000はNECと東芝が製造することになった。

1990年代の初め，MIPS社はSGIに吸収合併される。これには，MIPS社がSGIのGWS向けの高性能MPUを独占的に提供するという意図があったようだ。しかし，1998年，SGIはMIPS部門を切り離してしまう。きっかけは，R10000の次機種にあたる666MHz動作のBeast(コードネーム)の開発計画を断念したことに始まる。SGIのGWSは今後インテルのIA64アーキテクチャのMerced(コードネーム)以降のものに切り替える方針になった。R10000の現行品の改良版は開発するが，ハイエンドのMPUとしては新規アーキテクチャの製品を開発することはしない。MIPS社はより高性能な組み込み制御用のMPUやMPUコアの開発に力を注ぐことになった。

■NECの参入

1980年代の終わり，世の中のMPUがRISCという風潮になってきて，日本の半導体製造メーカーは，どこもかしこもRISCチップを共同で製造するパートナーを探していた。それまでオリジナル路線を標榜していたNECも例外ではなく，密かにMIPS社との接触を図っていたようだ。なぜ，MIPSだったか。当時，RISCとしては，MIPSだけでなく，SPARC，PA-RISC，POWERなどが勢力を伸ばしてきていたが，それぞれは，すでに別の半導体メーカーと契約を結んでおり，MIPSだけが残っていたというのはありうる話である。

NECとMIPS社の提携を最初にスクープしたのは米国のEE Timesという業界紙である。当初，NECは否定していたが，1989年にMIPS社との業務提携を正式に発表した。内容は，NECは当面，MIPS社のR3000を製造し，次期製品であるR4000を共同で開発する，というようなものだったと思う。NECのオリジナルMPUであるVシリーズとは，50MIPSの性能を境に，ハイエンドとローエンドで棲み分けるとされていた。もちろん，MIPS製品がハイエンドである。また，NECのオリジナルな半導体プロセスで製造されるMIPSのMPUはVRシリーズと名づけられることになった。R3000ならVR3000，R4000ならVR4000という具合である。

■VR3000

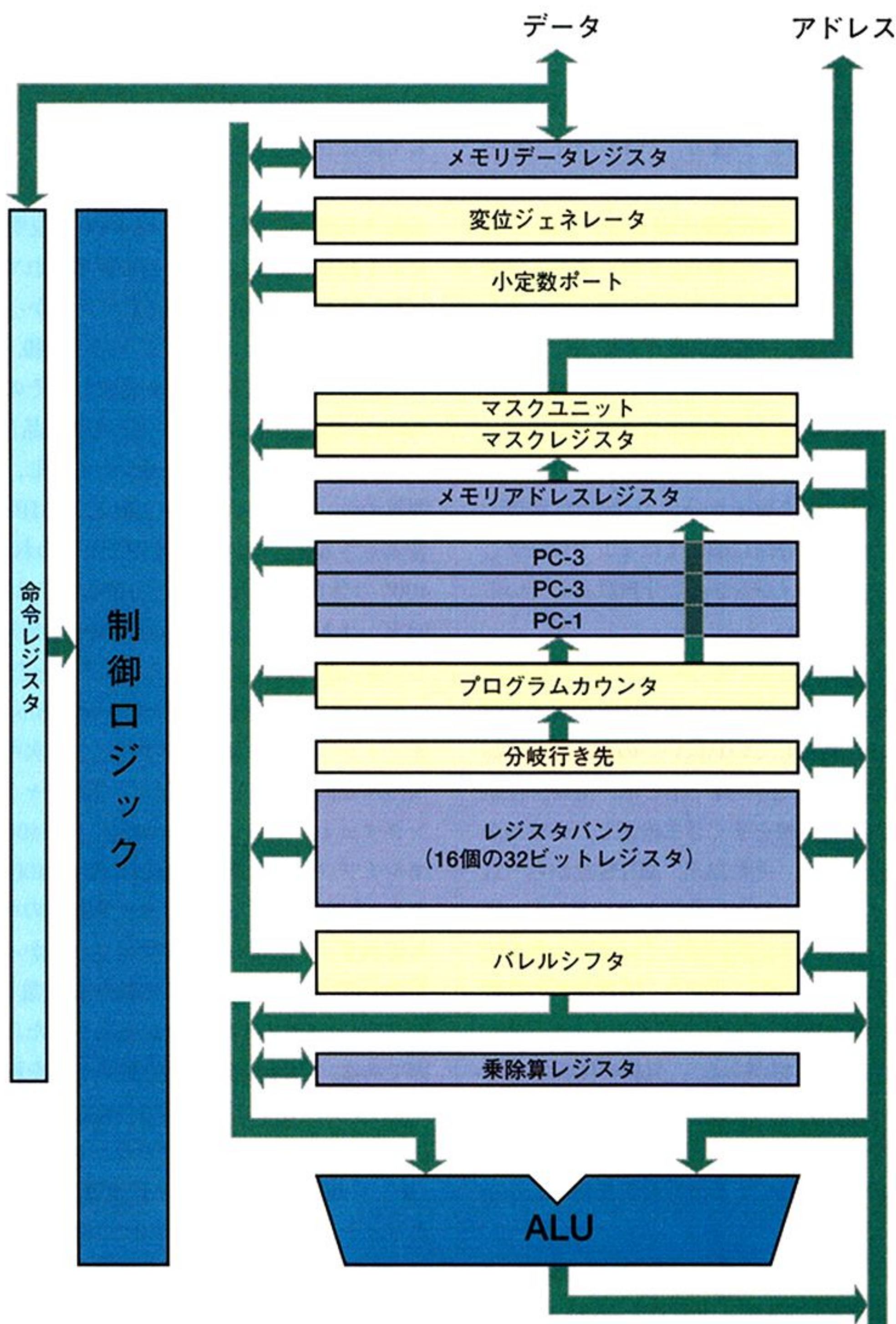
かくしてNECはVR3000を製造することになる。最初のVR3000は25MHz動作であったが，その高速版であるVR3000Aは40MHz動作である。25MHz程度の周波数なら当時のCISCのMPUと同等か少しだけ高速という程度なので，チップの製造も難しくはない。しかし，40MHzとなると簡単にはいかない。いろいろ工夫してみるのだが，その速度を実現することはできなかった。また，同じライセンシーであるIDTやPerformanceは，すでに40MHz動作を達成していたので，NECとしてもあきらめるわけにはいかなかった。万策尽きたNECが取った手段は，他社のプロセス技術をそのまま真似て製造するということだったという。噂ではあるが，こんな危険な製造プロセスではチップの寿命が5年も持たないとぼやきながら模倣したそうだ。かくして，NECも40MHz動作を達成する。NECとしては敗北だった。

しかし，そのままで終わるNECではない。独自にプロセス技術を進歩させていき，最近ではVR12000で300MHzという動作周波数を可能にした。しかし，インテルやDECはすでに500MHz以上の動作周波数を持つMPUを発表している。NECもそれらに追いつくために，日々，製造技術を向上させていることだろう。

図1　オリジナルMIPSプロセッサの構成

■VR4000/VR4400

VR3000の製造とほぼ同時期，VR4000の開発が始まっていた。VR4000とは，1次キャッシュ，整数演算ユニット，浮動小数点演算ユニット，2次キャッシュインタフェイスを備えた，100MHzで動作するMPUである。100MHzという高速で動作させるため，パイプラインの段数をVR3000の5段から8段に増やし，1パイプ当たり(単位クロック当たり)の論理を少なくした。MIPS社は

これをスーパーパイプラインと呼んでいるが，パイプラインの一変種にすぎない。現在の400MHz以上で動作するインテルなどのMPUでは10段以上のパイプラインは半ば常識になっている。

また，R4000でMIPS社はMIPSでないMPUを開発してしまった。つまり，パイプラインがインタロックするアーキテクチャになってしまったのだ。パイプラインの段数が増えたことにより，ロード遅延は1クロックから2クロックへ，分岐遅延は1クロックから3クロックに増えた。そして，ロードしたデータを2クロック以内に参照しようとする場合は，パイプラインにインタロックが生じて待ち合わせをするようになり，分岐は互換性を取るために分岐命令直後の1命令分だけを実行し，残りの2クロック分(2命令分)は抹殺するようになった。分岐に関してはインタロックしないが，パイプラインを2命令分無駄にしていることに変わりはない。

VR4000を開発していた当時のMIPS社の態度は，ライセンシーに対し，MIPS社が提供する製品をそのまま製造していればいいのだ，という高飛車な態度だったという。NECは，MIPS社からもらった回路情報をすぐさま検証し，いくつかのバグを指摘した。それ以来，MIPS社から一目置かれるようになったということだ。そして，回路設計やレイアウト技術に関しては本当の意味での共同作業が始まった。しかし，論理設計の分野では，まだまだ相手にされていなかった(現在も低く見られている感がある)。契約の関係もあったが，アーキテクチャレベルの質問には答えてはもらえなかったという。NECの社員はMIPS社から一方的に送られてくる技術資料を待つしかなかったらしい。

そうこうするうちに，開発は順調に(?)遅れ，1年が過ぎようとしていた。しびれを切らしたNECをなだめるため，MIPS社はV4000のRTL(論理レベルのHDL記述)を公開した。RTLには恐るべき秘密が隠されていた。R3000にFPUとキャッシュを内蔵した程度としか思われていなかったR4000は，実は世界初の64ビットプロセッサとして開発されていたのだ。

当時，MIPSアーキテクチャの解説書(英語)が市販されていたが，それのR4000の説明には64ビットMPUなどとはひと言も説明されていなかった。64ビット命令も掲載されていなかった。MIPS社の考えは，おそらくこうだ。最初，R3000の上位機種としてR4000を発表し，その翌年くらいに，R4000を64ビット化した別製品として発表するつもりだったに違いない。しかし，開発の遅れから，R4000を最初から64ビットMPUとして発表せざるをえなくなったものだと思われる。VR4000の発表は1991年だが，1992年にはDECの64ビットMPUであるAlpha(21064)が発表されている。世界最初ということにこだわったわけだ。

VR4000には3種類のパッケージがあった。2次キャッシュインタフェイスがなく，1次キャッシュのみで動作するVR4000PC，2次キャッシュインタフェイスを持つVR4000SC，VR4000SCにマルチプロセッサ機能を追加したVR4000MCである。しかし，マルチプロセッサ関係のバグは取り切れず，VR4000MCは販売されなかった。翌年の1992年にはVR4000の動作周波数を150MHzに向上させたVR4400が発表されたことも一因である。VR4400はVR4000のバグを取って高速化したMPUである。追加された機能は，内部状態の出力機能，マルチプロセッサ機能，マスタ/チェッカ構成によるエラー監視機能である。1次キャッシュの容量もVR4000の16Kバイト(命令8Kバイト，データ8Kバイト)から，2倍の32Kバイト(命令16Kバイト，データ16Kバイト)に増やされた。VR4400はVR4000と同じく，VR4400PC，VR4400SC，VR4400MCの3種類が開発された。また，1995年には動作周波数が250MHzの製品も発表されている。

■ VR3000系の展開

MIPS RISCは，もともとEWSの世界をターゲット市場にしていた。しかし，R4000クラスのMPUになると製造が難しく，脱落していくライセンシーが増えてきた(逆に東芝はR4000から参入してきたのだが)。そのようなメーカーはEWSとは違った世界に応用分野を求めようとした。ただし，パソコン市場はインテル一色でとても食い込めるものではなかった。彼らは必然的に組み込み制御分野に活路を見出すことになる。

1990年頃からR3000をCPUコアとしてキャッシュなどを内蔵し，組み込み制御分野に対応しようという動きが始まった。最初は，IDTが外付けキャッシュのパリティを削除したR3001を発表した。続けてIDTは，キャッシュとライトバッファを内蔵したR3051，R3052を発表した。キャッシュ容量は，R3051が命令に4Kバイト，データに2Kバイトであり，R3052はその倍の容量であった。同時期，LSI LOGICはキャッシュとDRAM制御機能を内蔵したLR3300を発表している。少し遅れてPerformanceはキャッシュ容量が，命令8Kバイト，データ8KバイトのPIPERというプロセッサを発表している。

従来は別チップであったFPUを内蔵しようという動きもあった。1991年，NECはVR3000とVR3010(FPU)を1チップに集積したVR3600を発表している。それに続いてPerformanceもR3400AというFPU内蔵品を発表した。翌年，IDTはFPUのほかにキャッシュも内蔵したR3081を発表した。キャッシュ容量は，命令16Kバイト，データ8Kバイトと，かなり大きい。NECもFPUとキャッシュを内蔵したプロセッサ(VR3700?)を開発しようとしていたらしい。が，これはなぜか発表されていない。すでにVR4000PCやVR4200が発表されていたので，必要ないと判断されたのだろうか。しかし，NECは，1993年には，FPUは内蔵しないものの，命令4Kバイト，データ1Kバイトのキャッシュとライトバッファを内蔵したVR3800を発表している。これは，LBP(Laser Beam Printer)に用途を限定していたが，VR4000を持っているメーカーが，なぜいまさらVR3000なのだろうか。NECのやることはよくわからない。RISCの開発部隊とは異なる部隊が独自に開発したからだという噂もある。3600，3700(?)，3800ときて，次は3900であるが，R3900は東芝が開発している。この技術は，のち(1996年)のTX39コアに引き継がれていると

表1 VRシリーズMPUの比較

プロセッサ	動作周波数 [MHz]	消費電力 [W]	プロセス [μm]	性能(公称値)		
VR3000A	40	4.0	0.8	35 MIPS		
VR3600	40	5.0	0.8			
VR3800	25	2.2	0.8			
VR4000	100	12.5	0.8	80 MIPS	59 SEPCint92	61 SPECfp92
VR4400	150	12.5	0.6		94 SPECint92	105 SPECfp92
VR4400	250	11.5	0.35		175 SPECint92	178 SPECfp92
VR4200	80	1.5	0.6		55 SPECint92	30 SPECfp92
VR4300	100	1.8	0.35	125 MIPS	60 SPECint92	45 SEPCfp92
VR4310	167	2.3	0.25	222 MIPS		
VR4305	67	1.5	0.35	106 MIPS		
VR5000	200	8.5	0.35		5.5 SPECint95	5.5 SPECfp95
VR5464	200	3.5	0.25	415 MIPS	8.0 SPECint95	3.6 SPECfp95
VR5432	167	2.5	0.25	347 MIPS		
VR1000	200	30.0	0.35		300 SPECint92	600 SEPCfp92
VR12000	300	24.0	0.25		16.8 SPECint95	27.8 SPECfp95
VR4100	DC~40	120m	0.35	45 MIPS		
VR4101	33	250m	0.35			
VR4102	66	250m	0.35	80 MIPS		
VR4111	100	180m	0.25	130 MIPS		
VR4121	168	N/A	0.25	224 MIPS		

思われる。

ところで，R3000が全盛期の頃，日経BP社の記者に聞いた話がある。東芝，IDT，LSI LOGICのマニュアルは英語版のみで，ユーザーサポートも悪かった。そのような状況のなか，NECは独自に日本語版マニュアルを作成していてサポートもいいので，他社の製品を使用している顧客から質問がたびたびきていたとか。しかし，時が流れ，時代がハイエンド指向になると，NECもハイエンド向けのプロセッサについては日本語版を作成するのやめてしまった。しかし，ローエンドの組み込み制御用のMPUについては，逆に，てねいなマニュアルが作成されるようになっていく。東芝もR3900シリーズからは日本語マニュアルを作成しているようである。

図2　スーパーパイプラインという言葉を生んだR4000の実行パイプライン

IF - IS - RF - EX - DF - DS - TC - WB
IF - IS - RF - EX - DF - DS - TC - WB
IF - IS - RF - EX - DF - DS - TC - WB
IF - IS - RF - EX - DF - DS - TC
IF - IS - RF - EX - DF - DS
IF - IS - RF - EX - DF
IF - IS - RF - EX
IF - IS - RF

■VR4400系の展開

VR4200はVR4400PCの廉価版として開発された。VR4400は，パッケージこそ，PC/SC/MCの3種類に分かれていたが，中身(ダイ)は同一のものだった。1次キャッシュのみの機能が必要なとき，2次キャッシュインタフェイスやマルチプロセッサ機能の占める部分が邪魔になる。つまり，ダイの面積が増えてコストが上がる。それならばということで，最初から1次キャッシュしかないVR4400PCを開発するという計画が持ち上がった。それが，VR4200である。発表は1993年，80MHzで動作した。開発にあたり，VR4400の8段パイプでは性能が出ないという理由で，パイプラインはVR3000と同じ5段に戻された(パイプライン段数は多ければよいというものではない)。動作周波数が80MHzと，比較的低かったのも5段パイプラインに戻した理由のひとつだろう。NECの発表では80MHzの動作でも100MHzのVR4000PC相当の性能が出るとしている。

さて，VR4200はVR4400とオブジェクト互換を主張していたが，アーキテクチャとしては異なる部分があった。たとえば，VR4400の非キャッシュライトは4クロック間隔なのに，VR4200では3クロック間隔に高速化してしまった。このため，従来の周辺回路が使用できなくなった。また，チップダイ面積を小さくするためにFPUを削除し，整数演算のデータパスを浮動小数点演算にも流用した。浮動小数点演算機能はなくなったわけではないが，性能は当然低下した。整数演算機能が第一で浮動小数点演算機能は二の次という組み込み制御分野にはそれでいいかもしれないが，VR4200はエントリレベルのEWSをターゲットに開発されたMPUだったので，不満が残るところである。マスコミへの発表によると，VR4200は，最初の1年間はNECが独占して製造し，そのあとはマルチソースとなるとされていたが，NEC以外に製造したメーカーは確認されていない。まあ，1年も経つと時代遅れのMPUになってしまいかねないから，しょうがないかもしれない。

Column　PlayStationのMPUはMIPS RISC

現在のPlayStationに搭載されているMPUはLSI LOGICがソニーと共同開発したものである。LSI LOGICはMIPSのライセンシーであり，当時(1993年頃)は，R3000をCPUコアとしたASICに注力していた。PlayStationのMPUの概要は34MHz動作のR3000をCPUコアにし，JPEG複合化回路(VDE)，3次元グラフィック処理回路(GTE：Geometry Transfer Engine)を1チップに集積していた。0.5μmルールのCMOSプロセスで製造され，CPU部は100トランジスタを集積していた。命令キャッシュは4Kバイト，データキャッシュは1Kバイトである。34MHzという比較的低速な動作であるが，各機能ブロックの並行動作により220MIPSの性能を出していた。独立したグラフィック処理ユニットはトータル性能を500MIPSまでに引き上げたという。

なぜ，ソニーはLSI LOGICを選んだか。『ソニーの革命児たち』によれば，Playstationの要求性能は800MIPSであり，回路規模はゲートアレイ換算で100万ゲートである。0.5μmプロセスで100ゲートのLSIが製造可能な半導体メーカは当時，世界に2〜3社しかなく，LSI LOGICがそのうちの1社だったという話である。LSI LOGICは100万個という要求数量に現実味を感じられず乗り気ではなかったが，開発費をソニーが負担するということで開発がスタートしたという。

当時の日本ではすでに0.35μmのプロセス技術が確立していたので，NECや東芝が聞いたら怒り出しそうな話である。ゲート規模も大したことはない。話を盛り上げるための脚色と受け取ったほうが無難である。

現在，巷では次期PlayStation(PS2)のMPUが何になるか話題になっている。米国の業界紙であるEE Timesのスクープによると，今年の2月にISSCCという学会でソニーと東芝が共同発表するMPUがPS2のMPUだという。同紙によると，MPUの概要は，128ビットのマルチメディア拡張をした250MHz動作のCPUコア，10個の浮動小数点積和算器，4個の浮動小数点除算器，MPEG2のデコーダ，10チャネルのDMAコントローラ等が128ビットの内部バスで結合されている。0.18μmプロセスで1億トランジスタを集積し，1.8Vで15ワット(!)の消費電力だそうである。CPUコアは2ウェイのスーパースカラで，16Kバイトの命令キャッシュと8Kバイトのデータキャッシュを内蔵し，100以上のマルチメディア拡張命令を実行可能だそうである。

ISSCCでは学会発表より前に，その内容を公表することを禁じているため，ソニーや東芝からはPS2のMPUについての正式なコメントはない。あるとしたら2月以降になる。

さて，東芝のMPUのロードマップには最近TX79というプロセッサが掲載されたが，これが件のISSCCで発表するMPUではないかとの憶測が飛んでいる。7という番号から，MPUの実態はQEDが開発したRM7000ではないかという見方もある。

ソニーの社長は「噂がひとり歩きしているだけで，学会発表と商品は別物」と発言しているが，真相はいかに。本誌が発行される頃にはすべてが明白になっているはず(これは1999年1月末時の原稿です。その後の状況につきましては「特別企画　PlayStation2の予備研究」をご覧ください)。

〈参考文献〉
1) 麻倉怜士,『ソニーの革命児たち』,(株)IDGコミュニケーションズ，1998年.
2) 浅見直樹,『ソニーのゲーム機用MPU，R3000やJPEG回路を内蔵』，日経エレクトロニクス1994年7月4日号,no.612,p16.
3) Anthony Cataldo. "Toshiba processor may be brains of next Sony Playstation",EE Times，1998/11/13.

同じ1993年には，QED社からR4600も発表されている。これも，VR4200と同じコンセプトを持ったプロセッサである。しかし，R4400の開発者が設計しただけあって，R4400との互換性はほぼ完璧だった。R4600でも非キャッシュのライトは高速化されたが，R4400との互換バスモードも備えていた。さらに，FPUも削除せず，たいした面積の増加にはならないとして内蔵されていた。R4600はIDTと東芝が製造していた。IDTがめずらしく日本語版マニュアルを作成していたので，そのMPUにかける意気込みが感じられたものだ。

1995年頃，マイクロソフトがWindowsNT4.0をリリースした。その時点ではMIPS系のMPUも対象としたOSだった。それを見たチップセットメーカーはMIPSのMPUでパソコン市場に参入しようと，新製品を開発した。有名なところでは台湾のACERが開発したPICAチップセット，メーカーは失念した（IDTだったか）が業界紙で大きく取り上げられたTigerSharkなどがある。NECもRabbitというチップセットを開発してパソコン市場に参入しようとした。Rabbitの特徴的なところはVLバスに対応し，PC/AT互換機のハードウェアをそのまま利用できるようになっていた点である。市販のPC基板に少量のハードウェアの追加だけで，VR4400PC/SC，VR4200をMPUとするパソコンができあがってしまうのだ。この時代にSDRAMインタフェイスを備えていたのも画期的といえるだろう。しかし，マイクロソフトがWindowsNT4.0の次からはMIPS系のMPUのサポートを行わないことをアナウンスしたため，Rabbitを初めとするMIPS用チップセットはどこかへ消えて行った。また，WindowsNTという応用分野を失った組み込み用途のMIPS系MPUは，各社こぞってWindowsCEの世界に流れていくことになる。

■VR10000/VR12000

1994年，R8000というMPUが発表された。これは，150MHzで動作する4チップ構成のMPU（といっていいのかな）である。内訳は，R8000 CPUコア，R8010FPU，キャッシュのタグRAM，キャッシュのデータRAMからなる。MIPSアーキテクチャとしては初めてスーパースカラを実現したMPUで，整数2並列，浮動小数点2並列の同時演算が可能になっている。公称性能は300 MIPS，300MFLOPS。動作周波数の2倍という，いかにもという性能だ。噂によると，このR8000はスタンフォード大学の大学院生の研究成果だという話である。担当教授はMIPSアーキテクチャの生みの親であるHenecyだという。

R10000というMPUは，端的にいうとR8000を1チップ化したものということができる。もちろん，最先端の技術が追加されていることはいうまでもない。具体的には，次の8点である。

- ・5並列の機能ユニット（整数×2，浮動小数点×2，ロード/ストア）によるスーパースカラ
- ・アウトオブオーダー命令発行
- ・レジスタリネーミング機能
- ・ノンブロッキングキャッシュ
- ・大容量1次キャッシュ内蔵（命令32Kバイト，データ32Kバイト）
- ・2次キャッシュとしてシンクロナスSRAMを接続可能
- ・分岐予測機構
- ・マルチプロセッサ機能

数年前，Henecy教授が来日して，これからのマイクロプロセッサについて講演をしていたが，いま思えば，このときに語られたMPUの必須機能はR10000の機能にほかならなかった。それだけ，R10000のアーキテクチャに自信を持っていたわけだ。私は未読だが，かの有名なHenecyとPattersonの共著である「コンピュータアーキテクチャ」の最新の版ではスーパースカラ技術が取り上げられ，R10000をモデルとしたと思われる仮想MPUが紹介されているという。

それはともかく，NECは1995年に，このR10000をVR10000として発表している。動作周波数は200MHzだった。さらにNECは，VR10000のアーキテクチャの改善と動作周波数を300MHzに高速化したVR12000を1998年に発表している。VR12000はレイアウト（回路配置）を全面的に見直して，高速化を実現したものである。アーキテクチャ的にも，アウトオブオーダー実行できる命令を溜めておくキュー（いわゆるリザベーションステーション）が32命令分だったのを48命令分に拡張したり，分岐予測テーブルを512エントリから2Kエントリに拡張している。

これらにより，VR10000に対して20%の性能向上になったということである。これは，NECのMIPS系MPUでは現在もっとも高性能なプロセッサであるといえる。MIPS社は，ハイエンド分野において新規アーキテクチャの開発を放棄したということであるが，NECはVR12000を基にシュリンクなどを行い，より高速なMPUを開発していくことだろう。NECのVRシリーズのロードマップにはハイエンドプロセッサとしてVR12000の次になにかがあると示唆している。もしかしたら，ハイエンドプロセッサも，あとで述べるVR4100と同様に，NECが独自に論理設計から開発する時代がくるのかもしれない。

■VR4300/VR4310

スーパーファミコンで成功を収めた任天堂が次機種に求めていたのはゲームのリアルタイム性だったという。スーパーファミコンの次機種での採用を当て込んで，任天堂にアプローチしてきた企業は「リアルタイム」ということがわかっていなかった。そのなかでSGIだけがリアルタイムの本質をわかっていた。こうして，任天堂とSGIは1993年にProject Reality構想を発表した。これまでの常識を打ち破るような3次元グラフィックを用いたゲーム機を，1994年にはアーケード版として，1995年には家庭用として開発するという内容だった。任天堂のイメージとしては，SGIの最高級のGWSであるOnyxを子供たちに与えることだった。設計の大部分をSGIが担当することになっていたので，当時SGIの傘下にあり，SGIもそのMPUを自社のEWSやGWSに採用していた，MIPS社のMPUが採用されることが最初から明らかにされていた。このMPUは0.35μmという微細プロセスを使用して製造される予定だった。当時，このプロセスが使用できるのは，MIPS陣営ではNECと東芝だけだった。莫大な額の売り上げが期待できるビジネスなだけに，NECと東芝の売り込み合戦は熾烈を極めたことであろう。噂によると，NECは採算を度外視した低価格攻勢に出て，この商談を勝ち取ったといわれているが，真相は定かではない。

Project Realityのために MIPS社が用意したMPUはR4300iというものだった。これは，ひと言でいうとVR4200の外部バス32ビット版だった。ただし，バスインタフェイスは，マルチプロセッサを意識したVR4000互換のもの（VR4200はマルチプロセッサ対応ではなかったが，バスはVR4000と同じだった）から，シングルプロセッサに適した単純なものに変更になった。

1995年，NECはR4300iをVR4300として発表した。その売りは1MIPS当たり28円というコストパフォーマンスだった。発表当時は，任天堂に採用されていることは秘密（現在ではインターネットのサーチエンジンでR4300をキーワードに検索するとNintendo64関係のサイトが何件もヒットする）みたいだったが，それを発表すると世間の受けももっと大きかったと思われる。しかし，応用分野のひとつにはゲーム機としっかり明記されていた。ハイエンドを信条としていたMIPS系MPUで応用分野にゲーム機が出てくることは珍しいことだったので，勘のいい人は気づいていたようだ。その後，1998年にはVR4300を0.25μmプロセスにシュリンクした167MHz版も発表されている（VR4300は100MHz動作）。VR4310も1MIPS当たり10円を割るコストパフォーマンスを売りにしている。もっとほかに特徴はないのかという感じだ。

ところで，このVR4310は2代目だそうである。NECの知人の話によると（真偽のほどは不明だ

が)，1997年頃，VR4300に対しマルチメディア系の命令を拡張したVR4320とVR4310というMPUを開発する計画があったという。おそらく，VR4320は外部バス64ビット，VR4310は外部バス32ビットの製品である。この設計はNECの米国の関連会社が担当した。しかし，開発は思うように進まず，事態を案じたNECはすべり止めとして，従来のVR4300のシュリンク版を計画した。はたして，VR4320/VR4310は完成に至らなかった。このため，予備のシュリンク版VR4300がVR4310として発表されたということである。能天気な開発者の中には，現在出荷されているVR4310が，彼らの開発したものだと勘違いしている人もいるらしい。

■VR5000

VR5000は，QEDが開発したVR4400のスーパースカラ版である。1996年にNECのVRシリーズのラインナップとして発表された。スーパースカラといっても，2命令同時実行で，これは整数演算と浮動小数点演算を同時実行する機能である。VR5000は，初めから200MHz動作を目標に回路設計がなされ，VR4400に対して1/5のコストダウンを実現している。プレスリリースによると，VR5000シリーズは，動作周波数の向上と，マルチプロセッサ機能のサポートを行う製品展開を予定している，となっている。しかし，その後，それらしきMPUが発表された形跡はない。VR5400という製品も発表されたが，これは2次キャッシュをサポートしていないという意味で，別の製品系列だといえる。1995年に発表のVR4400の250MHz品との差別化が難しそうである。NECのロードマップによるとVR4400はすっかり無視されていて，VR5400がVR5000の継承機種になっているが，どうもしっくりしない。

■VR5400

VR5400はSandCraftが開発したVR4200/VR4300のスーパースカラ版である。1998年に発表された。外部バス64ビットのVR5464と，外部バス32ビットのVR5432がある。その特徴は，2命令同時実行可能なシンメトリックスーパースカラおよび，分岐予測機能の採用などにより，VR5464は同じ周波数のVR5000より約50%，VR5432は同じ周波数のVR4310より約60%性能が向上していること，0.25μmプロセスによる回路設計により，消費電力の低減を実現していることなどである。セットトップボックスなどのマルチメディア機器，アーケードゲームなどのアミューズメント機器，カラーLBP/PPCが応用分野だという。VR5000とはピン互換で，2次キャッシュインタフェイスに相当する端子はノーコネになっているらしい。また，昔SGIが提唱していたマルチメディア命令セットであるMDMXとよく似たマルチメディア命令をサポートしている。マルチメディア命令をサポートするのはMIPS系のMPUでは最初である。

SandCraftというのは初耳の会社名であるが，わずか20余名の従業員で16カ月かからずにVR5400を開発したことを自慢しているようである。チップ製造の期間はこれに含まれてないはずなので，16カ月というのは時間をかけすぎという感がしないでもない。彼らの多くはR4300iの開発に関わっていたというから，VR5400はR4300iのモディファイ版のはず。実際，マイクロプロセッサレポートを読むとFPUは内蔵していないという(VR4200/VR4300のアーキテクチャだ)。案外，R4300iのデータパスを2重化して，分岐予測機能などを付け加えただけのものなのかもしれない。そうだとしたらSandCraftの実力は恐るるに足らず。415MIPSという性能も眉唾ものかもしれない。

SandCraftはVR5400とは別にSR1というプロセッサコアも発表している。VR5400と同じく，2ウェイのスーパースカラで，0.18μmプロセスで製造されたとき，400MHzで800MIPSの性能とか。世間では，このSR1を現時点の組み込み用途のプロセッサコアとしては最強と見ている人もいるが，はったりのような気がしてならない。NECがVR5400に社運をかけたりしないことを望む。

■WindowsCEへの道

1995年，マイクロソフトは携帯情報端末(PDA)やハンドヘルドPC用のOSとしてWindowsCEを発表した。その数年前からPegasusというWindowsCEの基になるOSのMPUと周辺ゲートアレイを基にしたプラットフォームが開発され

図3　SandCraft SR1コアの構成図

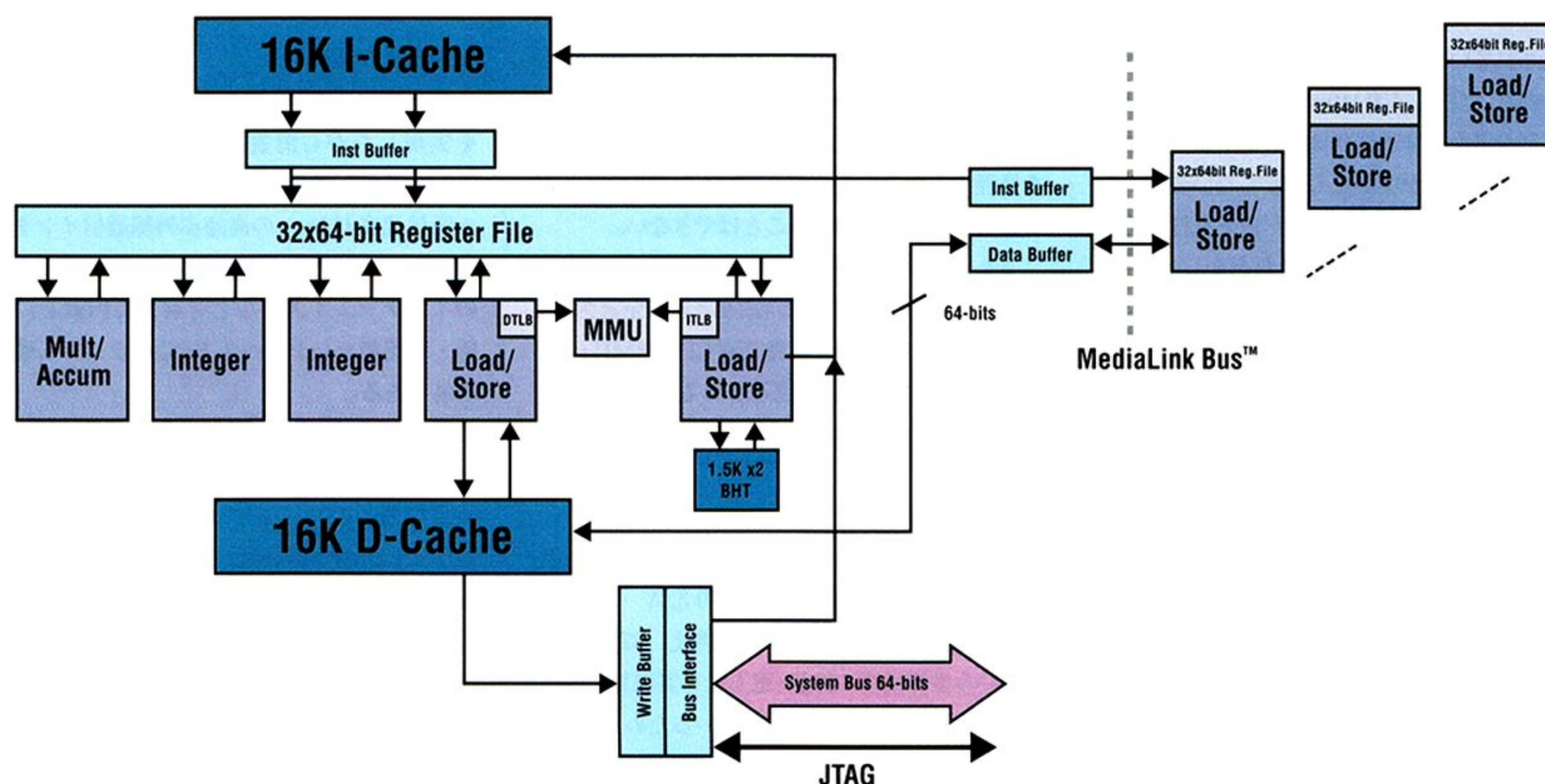
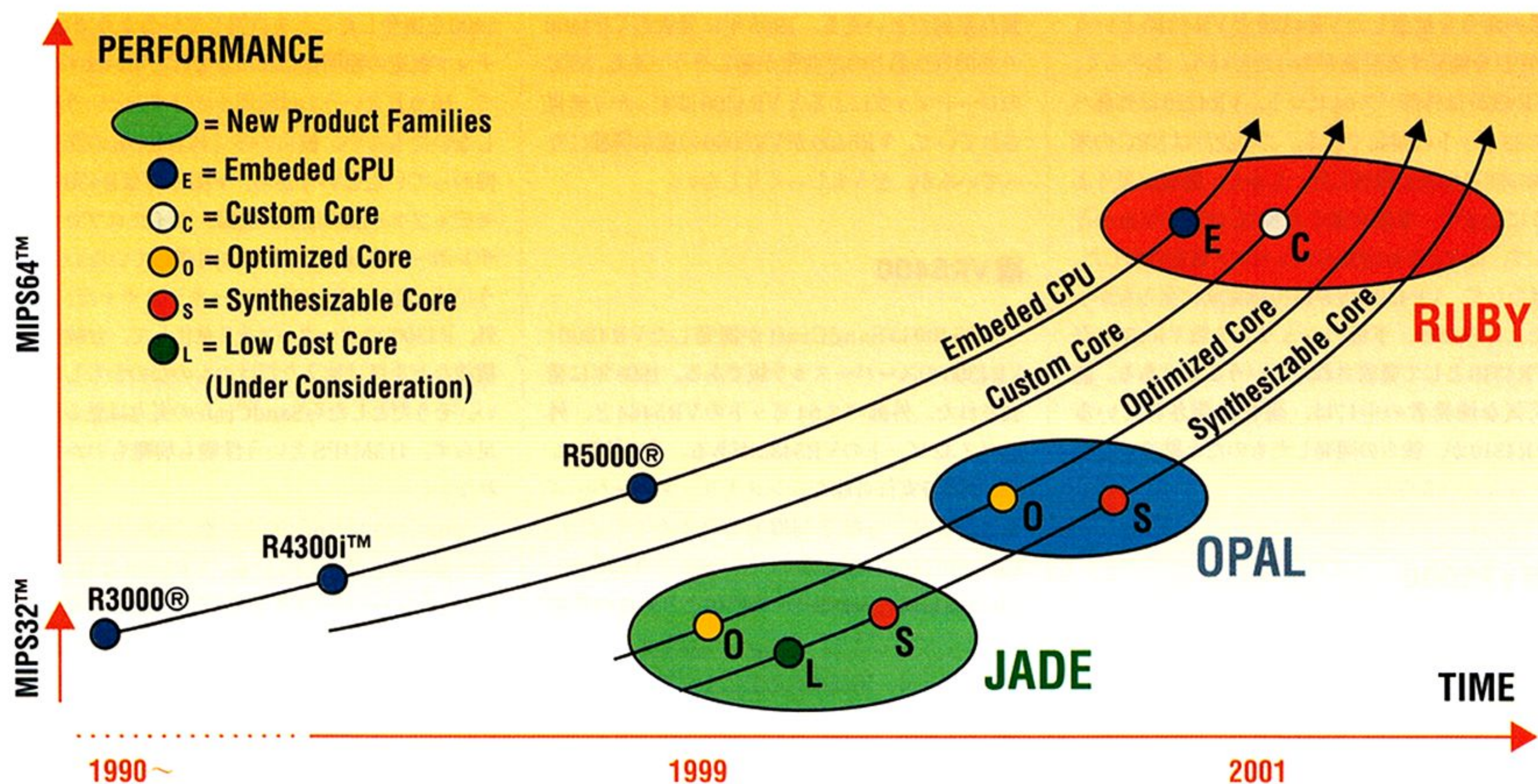



図4　MIPS Rシリーズのロードマップ

ていた。NECはそれにターゲットをあわせ，VR4100というMPUを開発し1995年に発表した。システムは電池駆動を必須としていたので，消費電力を低く抑えることが設計の第一目標であった。そのため，V810の開発で蓄えたノウハウをVR4200のRTLに適用することが試みられた。バスインタフェイスはVR4300と同じ仕様を採用した。とはいえ，NECにとっては初めてMIPS系のMPUをゼロから作り上げなければならなかった。それまでは，RTL記述や回路情報はMIPS社から供給されていたので，論理設計をする必要はなかったのだ。当然，VR4100の開発の初期には多くのバグが出た。マイクロソフトで用意したプラットフォームがなかなか立ち上がらないので，今度動かなかったらMIPS系のサポートを打ち切りx86に移行すると皮肉をいわれたとか。しかし，開発者の努力によりバグも収束し，VR4100はPegasusシステムでのMIPS系MPUの第1号に選ばれることになった。

その後，NECはVR4100をコアとして，1996年，1997年と立て続けにVR4101，VR4102というWindowsCEに必要な周辺機能を内蔵したMPUを発表した。2つのMPUの違いは動作周波数と内蔵キャッシュの容量で，VR4101は33MHz動作で，命令キャッシュ2Kバイト，データキャッシュ1Kバイト，VR4102は66MHz動作で，命令キャッシュ4Kバイト，データキャッシュ1Kバイトであった。また，1997年にマイクロソフトがWindowsCE2.0を発表すると同時にVR4111が発表されている。これは，OSの高機能化に対応するべく，動作周波数を100MHzに向上させ，キャッシュ容量を，命令16Kバイト，データ8Kバイトに強化したMPUである。

また，コンパクトな命令コードのために，LSI LOGICが提案していたMIPS16という命令セットもサポートしている。これは，命令コードのほとんどが16ビット長で構成された命令セットである。LSI LOGICではR3000ベースの32ビットMPUであるTR4101やTR4102(R番号がNECとかぶっている)にMIPS16をインプリメントしていた。64ビットMPUにインプリメントしたのはNECが最初である。MIPS16には64ビットのデータを処理する命令もあるが，TR4101/TR4102ではそれらの命令を使用することはできない。

さらに，WindowsCEのカラー化に対応するべく，より高性能なVR4121が1998年に発表されている。CPUコアを新たに作り直し，168MHzという高い動作周波数を実現している。それぞれのMPUの特徴をプレスリリースから拾ってみよう。

● VR4100

＊1ワット当たりで世界最高の性能を有するCPUコアときめ細かな動作制御により3.3V・40MHz動作時の消費電力を120mWに抑え，375MIPS/Wという世界最高の性能/電力比を実現している。
＊積和演算ユニットを搭載しているため，手書き文字認識やモデム処理などをソフトウェアのみで実現できる。

● VR4101

＊CPUコアにMIPSアーキテクチャに基づきNECが独自開発した，高性能/低消費電力の64ビットRISC型マイクロプロセッサVR4100を採用し，高速処理を実現している。
＊メモリコントローラやLCDインタフェイス，通信インタフェイスをはじめ携帯型情報機器に必須の周辺機能を内蔵しているため，小型で高性能な携帯型情報機器を容易に実現できる。

● VR4102

＊66MHz，80MIPSという従来製品の2倍の性能を実現しながら消費電力を従来製品と同等の250mWに抑え，高い電力/性能比を実現している。
＊通信速度4Mbpsの高速赤外線通信インタフェイス，ソフトウェアでモデム機能を実現するためのインタフェイス，などを新たに内蔵し，高性能・多機能なハンドヘルドPCのシステム構築が容易である。

● VR4111

＊最先端の0.25μmプロセスの採用により130MIPS(100MHz動作時)という高速処理と，内部論理動作部分の2.5V化により180mWという超低消費電力化をともに実現し，消費電力当たりの性能を従来製品の2倍以上に向上させている。
＊16ビット長命令セットも実行できるため，メモリの使用効率が大幅に向上し，プログラムサイズを従来と比べて最大で約40%縮小可能である。

＊VR4102とソフトウェアおよび端子レベルでのハードウェアの互換性を有しており，新製品への移行が容易である。

● VR4121

＊CPUコアにMIPSアーキテクチャに基づきNECが独自開発した高性能/低消費電力の64ビットRISC型マイクロプロセッサ・コアVR4120を採用し，168MHz動作という高速処理を実現している。
＊EDO-DRAMに加え，SDRAMにも対応したことにより，より高速なメモリシステムを構築できる。
＊従来製品との完全な上位互換性を有する端子配置を採用しMPUのアップグレードが容易となっている。
＊H/PCに必要な周辺機能を1チップに集積しているため，セットを短期間で開発できる。

このように，VR41xxシリーズは上位互換性と，WindowsCEに特化した周辺機能を1チップに内蔵していることをアピールし続けてきている。その甲斐があってか，現在，MIPS系のWindowsCEのプラットフォームはほとんどがNEC製のMPUという状況になっている。

このWindowsCEの分野でさらに高性能なMPUを開発するキーポイントはなんだろうか。WindowsCEのシステムに使用されているDRAMの動作周波数はEDO-DRAMで33MHz程度，SDRAMで66MHz程度であり，CPUの動作スピードに比べて非常に遅い。つまり，バスネックである。これを解決しなければならない。パソコンの世界ではPentiumやCeleronが巨大な2次キャッシュを内蔵してバスネックによる性能低下を低減している。WindowsCEのような携帯端末では2次キャッシュというのはあまり考えられない。

ひとつ考えられるのは命令ストリーミングという機能である。これはR3000特有の機能で，命令キャッシュがミスした場合に，命令をキャッシュに取り込みながら同時に実行するというものである。命令ストリーミングはバスネックな環境下で特に威力を発揮する。ところが，なぜか，R4000系以降の64ビットMIPSアーキテクチャのMPUでは命令ストリーミング機能がない。キャッシュに命令が格納されて後に命令の実行を始めるようになっている。VR4121の次機種(があるとしての話だが)では命令ストリーミング機能を採用したら面白いのでないか。さらに性能を向上させるためには，分岐予測機能，ノンブロッキングキャッシュ，スーパースカラといった機能もあり，これらは順次採用されていくことだろう(本当かなあ)。いま，CPUではWindowsCEの世界が熱い。

■NECとMIPSの今後

MIPS社はハイエンドの分野をあきらめて，高性能なCPUコアの商売を始めたようだ。現在，判明しているCPUコアにはJadeとOpalとRubyがある。

Jadeコアは，SOS(System On Silicon)のために設計された，高性能/低消費電力の32ビットMIPS RISCコアである。命令セットはR3000にR4000のTLBと特権命令を追加したものである。最大周波数は150MHz。1次キャッシュの容量は，命令，データ別々に2K～16Kバイトの間で選択可能で，デフォルトは，命令，データとも8Kバイトずつである。5段パイプラインを使用し，ほとんどの命令は1クロックで実行される。ソフトウェアモデム等の実現に必要な積和演算も1クロックで実行できる。0.25μmプロセスで使用するとき12平方mmのダイサイズという。

Opalコアは64ビットのコアで，R5000クラスの性能を持つということくらいしか公表されていない。

Rubyコアは64ビットのCPUコアで，周波数が500M～666MHzで1000MIPS以上の性能を見込んでいる。MDMXという，グラフィック性能を強化するマルチメディア命令を含んだ新命令セットに準拠し，将来は1GHzという超高速動作を目指す。応用分野は，セットトップボックス，アミューズメント機器，インターネット家電といった，21世紀に向け高度化するデジタルコンシューマ市場を睨んでいる。

1999年の1月に，NECと東芝はMIPS社とRubyのライセンス契約を結んだ。NECとしては，1000MIPS以上の性能を持つRubyコアを使用した第1弾を2000年末までに出荷する予定だとか。東芝も2001年に製品化を予定しているようである。

NECは，今回のライセンス契約を契機としてMIPS社との関係をよりいっそう強化し，今後とも高性能かつ高コストパフォーマンスのマイクロプロセッサを提供することを目的として，10年間にわたるMIPS社とのRISC型マイクロプロセッサに関する契約を結んだ。少なくともあと10年はMIPS系のMPUが滅びることはなさそうである。

■おわりに

VRシリーズの過去から現在までを俯瞰してきた。MIPS RISCの話ということで，今巷でPS2のMPUと噂されている東芝のTX79にも触れたいところだが，VRシリーズがメインということで，今回は割愛する。世間ではMIPS社の将来について必要以上に不安を感じている人も多いが，NECや東芝という優秀なパートナーがいる限り，未来は明るいのではないだろうか。

Column 最近の動向より

去る3月9日，SGIからハイエンドのMIPSプロセッサについて，ロードマップが発表された。2000年に，R12000の高速版(410M～450MHz)のR14000を，2001年に，命令キャッシュとデータキャッシュをそれぞれ64Kバイトに増加し，さらに高速化(600M～800MHz)したR16000を，そのあとにR16000のシュリンク版であるR18000を出荷するという。

どれも，先の公約(?)どおり，R10000の改良品にすぎないが，この時期にロードマップを発表するとは，インテルのMercedの開発が遅れているためであろう。R14000はすでにNECが開発中ということである(NECだからVR14000か)。NECとハイエンド分野で競いあっていた東芝はPlayStation2と組み込み制御の分野へ行ってしまったし，NECはこの分野でもまだまだ活躍していくのではないか。それにしても，R18000までが登場する事態になったら，そのときはMercedが失敗していることを意味するのではなかろうか。

3月22日，シャープから，シャープ製としては国内初のWindows CE機であるTeliosが発売された。使用されているMPUは東芝製のMIPS RISCであるR3922(129MHz)である。このTeliosは性能面でそれまでトップを誇っていたNECのMobile GearIIをあっさり抜き去った。最新のMobile Gear IIに使用されているMPUはNEC製のVR4121(131MHz)である。Teliosの性能を凌駕するマシンとして期待されているのは，まだ発売の噂しかないが，日立のSH4(128MHz?)を使用したPERSONAである。しかし，筆者はNECの次期WindowsCE機に期待したい。国内のWindowsCE機では圧倒的なシェアを持つNECがこのまま黙っているはずがない。どのようなマシンが出てくるか楽しみである。

《参考文献？》
1）Stephen B. Furber,『比較研究 RISCアーキテクチャ』，日経BP社，1992年．
2）神保進一,『最新マイクロプロセサテクノロジ』，日経BP社,1996年．
3）矢田真理,『ゲーム立国の未来像』，日経BP社，1996年．
4）"NEC VR5400 Makes Media Debut"，MICROPROCESSOR REPORT，Vol.12，No.3，1998．

《参考URL》
1）http://www.ipc.nec.co.jp/japanese/today/newsrel/
2）http://www.sandcraft.com/
3）http://www.mips.com/

御託僕鐸 annex 2

建前と現実との間で

小笠原陽介 Ogasawara Yousuke

CPUをオーバークロックするということ

パソコンに限った話なのかどうかわからないのだが，最近私は，情報によって私たちの気持ちや考え方というのが，どのようにコントロールされてしまっているか，ということに興味を持っている。いや，コントロールされて，などというのは正しくないのかもしれない。別に誰もコントロールしようと思っているわけではないからだ。というより，その情報を発信する側までもが，意図しないそのような効果に戸惑う場面が多いのではないかと思っている。

別に私はそのようなことを分析している研究者というわけではないし，抽象的な話をするのが目的でもないから，具体的な話を出してしまったほうがよいのだろう。私が常々「こりゃ変だぞ」と思っているのは，CPUのオーバークロック使用についての話題なのである。

CPUというのは，いずれも「××MHzで動作します」という保証の下で販売されている。しかし，たとえば300MHz用のものが，きっかり300MHzでしか動作しないかといえば，必ずしもそうではない。なぜかといえば，「きちんと300MHzまででしか動作しない」ものを作るのは非常に難しいことだからだ。ある目標値に対して，誤差の範囲を狭めることは可能だとしても，実際の結果は，ミクロに見れば必ずバラつきを生じるからである。ついでにいえば，このバラつきは，おおむね正規分布をなすだろう。したがって，最初からきっかり300MHzを目標に作ったとすると，製造時の歩留まりはほぼ50%になってしまうことになる(300MHz以上で動くものとして)。これではあまりにも効率が悪い。

300MHzで動作する製品を作りたければ，最初から多少のマージンを設定して，たとえば330MHzを目標にしておく，というようなことをする(マージンの数字に根拠はない。念のため)。そうすれば，正規分布の下限にあたる，つまりは多少デキの悪いものでも，300MHzでならば十分に動くというわけだ。上限側のものなら360MHzで動くかもしれない。

もちろん，それは「運がよければ」可能，ということにすぎなかった。また，オーバークロックで動作させた場合には，ハードウェア的にもソフトウェア的にも，さまざまな問題を生じる可能性もある。過熱による発煙・発火などの可能性すらないとはいえない。メーカー保証外の使い方である以上，それらを「自分の責任で」引き受ける覚悟のある人しかやってはいけないことなのである。

オーバークロック利用の広まりと奇妙な勘違いの発生

CPUをオーバークロックで利用する，ということ自体は，以前からある程度可能であったし，知っている人は知っていた。ただ，以前は，クロック周波数をあまり細かく調整できないとか，オーバークロック動作させられる幅が小さいとかいった理由があって，さほど一般的とはいえなかった。実際，危険を冒してオーバークロック動作させても，メリットはあまりない，ということが多かったのも事実だ。

ところが，Pentiumプロセッサ(P54C)が一般的になってきた頃から，いろいろと状況が変わってきた。ベースクロックをチップ内部で何倍にして動作するか，ということが信号によって制御できるようになり，マザーボードもベースクロックを切り替えられるようになった。P54Cチップ自体も，オーバークロック耐性が比較的高かったといってよいだろう。こうした事情から，オーバークロック利用ということ自体が，徐々に広まっていったように思う。

しかし，こうした傾向にもっとも拍車をかけたのは，Slot 1用のCPUだっただろう。どれがどう，ということはいわないが，ものによって，保証値よりも「いくぶん高い」どころではなく，相当高い周波数で動作してしまうケースがあることが知られるようになったのだ。

そうなる理由はいろいろあって，一概にはいえない。設計上の余裕はあるが，製造時の精度や歩留まりの問題から，低めの保証となっている場合もあるだろう。あるいはまた，設計上も製造段階もほとんど問題はないのだが，市場投入の戦略から，意図的に低い数字を提示している場合もありうる。真相はわからない。

理由はどうあれ，300MHz用と銘打たれたものが，400MHzとか，それを超える周波数で動作するとなれば，誰しも魅力的に感じてしまうだろう。パソコン雑誌なども「あくまで自己責任で利用するなら」と前置きしつつ，こうした事実にたびたび言及した。

ただ，同じCPUであっても，すべてがオーバークロックで利用できるとは限らない。もちろん個別差もあるが，製造ロットの違いによっても大まかな傾向はある。その帰結として「このロットならどのくらいまでクロックアップできる」といったことが関心の対象になる。もちろん，こうしたことは個人間で話されている分には問題ないし，パソコン雑誌でも，ある程度は許容せざるをえないことではあると思う。

しかし，こういう話題があまりにも流布しすぎた結果，妙なことが増えてきた。インターネットのWebページや，Web上の掲示板などを見ていると，オーバークロックでの利用が当然であり，それを試みないのは普通ではない，あるいは，オーバークロックで利用できなかったから損をした，というような書き込みが多く見受けられるようになってきたことだ。

Webで発言しているだけならまだよい。聞いた話では，メーカーやパーツショップに「これはどこまでクロックアップできますか」と真顔で尋ねる人や，さらには，オーバークロックで利用できなかったり，できても小幅だったりすると，文句をいったり，返品を申し出る人までいるのだそうだ。こうなると，完全な勘違いとしかいいようがない。

情報の著しい流通が抜け駆けを加速させる構図

このような勘違いが生じるのはなぜなのだろうか。単に，オーバークロック利用というものの前提が，十分に知られていないだけなのだろうか。あるいは，そうした人が，個人的に，物の道理をわきまえていないだけなのだろうか。実をいえば，私も最近まで，これらのいずれかだろうと思っていた。

しかし，このところ，ちょっと考えが変わってきた。確かに，そうした個人について見ると，情報不足なのかなと思うこともあるし，いくぶん節操のない利己主義のにおいがすることもある。だが，もっとマクロな視野で見たとき，より本質的

に指摘されるべきは，情報化社会の中での「建前」と「本音」とのダブルスタンダード状況が，必然的にそういう人なり，そういう言動なりを生み出す，ということではないかと思ったのだ。

実はこれは，社会学者の宮台真司氏(都立大学助教授)が，その著書(*1)の中で，いわゆる「援助交際」について述べていた内容にヒントを得ている。援助交際とパソコンのクロックアップとでは，ずいぶん違うじゃないか，と思うかもしれないが，現象を生み出す構造は，不思議なほどに符合するのだ。

同氏の論を引用しているとキリがないので，ごくごく簡単に要約すると，こういうことだ。世間には「売春はいけない」という建前があり，同時に「それを多くの人が信じているはずだ」という信頼がある。しかし，一方で，日本は世界に名だたる買春天国であるという「現実」がある。この両者の乖離は，かつては隠蔽されていたはずだった。しかし，1980年代半ば以降，テレクラやQ²ダイヤルなどを通じて，実際に買春を持ちかけてくる男たちと出会うことで，女子高校生たちは「本音」を知るようになる。さらに，新聞報道や雑誌記事などによって，「世間の女子高校生はこんなことをしている」というマスイメージが広がり，「なんだ，みんなやってるんじゃないか」というかたちで，抜け駆けは促進される。こうして，結果として，多くの「普通の子」が援助交際に参入してくる，ということである。

では，こうした構図が，パソコン用CPUのオーバークロック利用の場合と，どのように似ているのだろうか。両者に共通するキーワードは「抜け駆け」である。以下，私なりに考えたことを述べていく。

チップメーカーによるクロック保証値というのは，いわば「建前」である。そして，パソコンというものは，それぞれに搭載されたCPUが保証している数十パーセントの速度差によって，かなりの価格差がつけられているものだった。こうした価格差は，この「建前」が生きているからこそ納得できるものだった。

従来は多くの人が，この建前を共有していた。大手パソコンメーカーは当然のこととして，中小のメーカー，それにパーツを組み合わせてショップメイドのマシンを販売しているところに至るまで，表向きは，まずこの建前から出ることはない。たとえ，マザーボード上のジャンパをちょっと切り替えるだけで，上位機種と同じスペックになったとしてもである。

オーバークロック動作ということが，さほど広く知られていなかった頃は，それはユーザーの中でも「建前」に反する「抜け駆け」であり，アンダーグラウンドな事柄として，ひっそりと語られていたことにすぎない。また，その実際は，技術的な知識ないし技量を持つ一部の人々が，特権的に享受できるメリットでもあった。当然ながら，そうした敷居の高さは，一般の人々を近寄せず，したがって「建前」それ自体の存続を脅かすものではなかったのである。

ところが，情報の流通が，この図式を破壊してしまった。その気になれば，そしてよほど運が悪くなければ，オーバークロック動作をさせることができるという「本音」がわかってしまったときに「建前」の持つ説得力は，大幅に減殺されてしまったのである。さらに，そうしたオーバークロック動作という「抜け駆け」を，実際に多くの人が行っているという「現実」が広く知られるようになると「なんだ，みんなやってるんじゃないか」というかたちで，抜け駆けは促進されるのだ。

もちろん，現実にクロックアップ手段が容易だったことが，この背景にあったことは言うまでもない。そうでなければ，多くの人々がクロックアップに手を出すこともなかっただろう。ただ，ここで重要なのは，それは主たる要因ではないということだ。また「建前」と「本音」または「現実」との間に乖離がある，という事態そのものが主要因でもない。そうした乖離や，「抜け駆け」をする人の存在が，「広く知られるようになったこと」が最大の要因なのだ。

建前と現実との間で ほどほどに折りあうこと

では，こうした状況に対して，どのような処方箋が考えられるだろうか。

前項で挙げた，宮台氏の援助交際についての指摘の中では，もはや「建前」が空虚化してしまっていることと，そうした「建前」をなおも押しつけようとする倫理的規制の無効さが指摘されたうえで，「現実」に対応した社会的信頼の回復には，個々人の自己責任原則を貫徹することが必要だ，としている。

ただ，これはCPUのオーバークロックについていえば，いくぶんわけが違う。チップメーカーによる動作クロック保証値というのは，製品についての公式な約束であり，引き続き必要かつ有効なものであり続けるからだ。前にも述べたように，オーバークロックで動作させた場合には，ハードウェア的またはソフトウェア的に，さまざまな問題を生じる可能性がある。

しかし，だからといってメーカーが，オーバークロック動作を防ぐようなリミッタを入れることも，あまり賢いとは思えない。そのような措置は，ごく短期的には動作クロック保証値による価格序列の維持に役立つとしても，製品のコストを押し上げるために，中・長期的に見れば，メーカー自身の足かせになるだろう。もちろん，あらゆるスパンで見て，ユーザーにとっての利益にならないことはいうまでもない。

結局，落ち着きどころとしては，やはり自己責任原則の貫徹，ということになるだろう。オーバークロック利用という現実があることは隠蔽する必要はないとしても，しかしパソコン雑誌などがそれをあと押しするような書き方は，やはりするべきではない。一般論として，そのような使い方の可能性を提示することはあってもよいが，どのチップ，どのロットが，どの程度まで，といったことを書くのは避けるべきだ，と私は考える。それはあくまで，購入したユーザーが自己責任に基づいて試せばよいことだからである。

建前と現実は，いつも一体ではない。だが，誰もが抜け駆けすることばかりを考えるようになることも好ましくない。私たちは，建前と現実の間で，ほどほどに折りあっていく知恵を身につけるべきなのだ。

*1 宮台真司「まぼろしの郊外　成熟社会を生きる若者たちの行方」(朝日新聞社，1997年)

後日談 『やってる側の人の話とか』

O：DOS/V magazineの記事とかではよくこういうの書いてますよね。

U：半分本職ですから。私も少しはやりました。/V magの場合は一般ユーザーができないことをやってみせるという面もありますので。まあ，秋葉原では結構文句いわれてるみたいですね。「冷凍車持ち出すことはないだろ」とか。ただ，WinBenchと3D WinBench，あとWinstoneの一部も取ることが多いんで，それくらいは動く設定にします。まあ，この辺が動けばたいていのものは動くんですが。

O：動かないものもあるということですか？

U：Windowsが立ち上がって普通に使えているようでも，クロック下げないと動かないプログラムはありますね。低い負荷でしかチェックしてないといきなり凍ります。

O：それでもクロックアップはすると？

U：そうでもないんですが，新しいものがあれば一応上げてはみます。逆に私が使ってるマシンで300Aをベース66MHzでドライブしてると「え？」という顔をされますね。

O：実際，意外なんですけど？

U：ちゃんとしたクーラー買ってないんです。リテール版のクーラーのままオーバークロックしようと考えるのは馬鹿げてます。

O：やっぱり冷やせば回りますか？

U：冷やさないよりは。CPUmarkとか取ってるとプログレスバーが3割くらいになったところとか，負荷が高いとこで猛然と団扇で扇いだりしましたね。600MHzとかいってやってたときは，Windowsの起動ロゴのところで炭酸ガス全開で吹きつけると動いたりしてましたし。

O：普段はどんなクロックで使ってますか？

U：メインマシンのK6は75×3の225MHz(233MHzの石ですけど)。PentiumODPはそのまま166MHz，Celeron300Aは300MHzですね。ついでにいえば，ノートのDX4/100は50MHzで使ってます。そのほうが電池のもちがいいんで。もひとつついでに68030は33MHzの石を35MHzで動かしてます。

O：ほとんど上げてませんね。

U：クロックアップということでいえばX68030のCRTCをいちばん上げてますね。80MHzですけど，SX用にヒートシンク貼って。

O：CRTCですか？

U：画面が広くなるんです。CPUクロック上げたってたいした違いはないですが，これは効果が絶大ですし。けど，修理に出せないんで，本当にOwnRiskですね。

仏日翻訳プログラム「来来仏語」

石上 達也 Ishigami Tatsuya

復刊号で紹介したフランス語→日本語翻訳プログラムがバージョンアップしました。より正確な解析を求めて，日々改良されていきます。自然言語処理はまだまだ課題の多い分野です。それだけにアマチュアプログラマにも楽しみの残っているテーマといえるでしょう。

そんなわけで，私がいまだに「Gran' Tourismo」でA級ライセンスが取れずにもがいている石上です。PlayStationというと，一般消費者向けと思い込みがどこかあり，少し軽く見ていたんですが，これがなかなか難しい。難しいけど，面白い。いまだに，A級ライセンスが取得できません。

という話を丹さんにしたら，

「うん，私もね，完成したあと，ちょっと(ゲームから)遠ざかってたら，いつもゴールドプライズを出せるわけではなくなっちゃいましたよ」

といわれてしまいました(とほほ)。A級ライセンスが取れないということは，レース場の半分も走ったことがないということですからね，なんとしてもゲーム代の元は取らなければいけません。

ま，このゲームのおかげで，カーブ前の減速がいかに重要かということがわかったので，それだけでも元は取れているんですけどね(机の上にAvisから1055ドルの請求書がすでにあったりして。今度からはFR車を借りよう……)。

さて，復活「Oh!X」もおかげさまをもちまして，なんとか2号を迎えることができたようです。

その場合、パソコンの翻訳結果を，3分の1は「てにをは」を変更する程度の修正で使え，3分の1は語順を入れ替える程度の修正で使え，3分の1だけが大幅な書き直しというレベルまで引き上げることが条件である

(大場五夫「パソコン翻訳入門<翻訳ツールの上手な使い方>」より)

いままで自動翻訳の参考書というと，正直なところ，教科書的なものばかりであまり有用なものという気がしませんでした。そんななかで，参考文献1は翻訳プログラムの作成者ではなく，プロの翻訳家が不完全な自動翻訳プログラムをいかに実際に使うかを具体的に示した貴重な参考書です。

実は，

I go to school.　　私は学校へ行く。
I come to school.　私は学校に来る。

のように，前置詞は動詞との組み合わせで訳語を決定すると，それなりに自然な日本語を出せることを「来来仏語」作成中に発見し，これはわりと画期的なことではないかと思っていました。しかし，この本の中に，ずばり〝生き残る翻訳ソフトの条件〟という項目があり，

「特に動詞の登録においては，前置詞を伴う型の登録も可能にすること。その場合，使用できる前置詞を制限しないこと。

<例> device 目的語1into 目的語2」

と書かれていました。

また，一見して自動翻訳が無理そうな文章を見つけたら，

From the appropriate directory under OpenWindows, type ocr. The following window will appear.
→ From"対応するディレクトリ" under"オープンウィンドウ", type "ocr" The "次のウィンドウ" will appear.

と前処理して，括弧内(〝～〟)をすべて名詞扱いすれば，意味論的な成果はともかく，翻訳結果はそれなりのものが得られる，というように，自動翻訳の正攻法だけに頼らず，ちょっとした小技をふくめ，よい日本語訳を得るためのノウハウが紹介されています(「来来仏語」では，まだ，ユーザーインタフェイスにまで手が回りませんが，将来はぜひサポートしたい機能です)。

さて，この本の筆者は，50ページのマニュアルを翻訳する場合の例を挙げて，上記の条件がクリアされていれば，

1日目	OCR(原稿をスキャナで読み取ること)
2日目午前	OCR(続き)
2日目午後	スペルチェック
2日目夜	自動翻訳(無人運転)
3～4日目	手直し

で，翻訳が行われると試算しています。当時の相場として，50ページのマニュアルの翻訳は，だいたい納期が5日だそうですので，コンピュータの活用によって，1日分，まるまる納期が短縮できることを示しています。

最初から志を高く持ちすぎても，あとで身動きが取れなくなるだけですので，とりあえず，大橋氏の提唱する最低条件をクリアすることを「来来仏語」の目標にします。

この本は，初版から3年も経つのに，例として使用された翻訳結果は，最新の翻訳プログラムでもそんなに違わないように思えます。ここ数年，翻訳エンジンそのものの性能はほとんど上がらず，GUIなどの派手な部分にばかり労力が注がれている，という筆者の意見に私も賛成です。英日翻訳に限っていえば，新規参入もひと段落し，インターネット対応もひととおり終わったようですので，今後は翻訳エンジンの改良を積極的に行ってくれることを期待しましょう。

プログラムの内部構成

今月は「来来仏語」の内部構成を少し説明します。プログラム自身がまだまだ完成していないことと，フランス語という特殊な言語を用いているため，あまり詳細には立ち入らず，英語など他言語の自動翻訳を行う際にも必要となるだろう個所を中心に説明します。

●単語の切り取り(形態素分析)

「来来仏語」の基本構成は，class TOKENです。これは，ほぼフランス語の単語一語に対し，ひとつのTOKENが該当します。

「来来仏語」に入力されたフランス語は，まず，単語ごとに切り離され，メモリ内にTOKENとして記憶され，翻訳にかけられます。この作業は，日本語のひらがな→漢字変換など，明確な区切り記号がない場合はとても大変です。

例) かれがくるまでまつ。
・かれが/くるま/で/まつ(彼が車で待つ)
・かれが/くる/まで/まつ(彼が来るまで待つ)

ただし，フランス語には，明示的な区切り記号があり，それも以下の3種

類しかありませんので，この作業は簡単です。

1）空白文字で切り離す

Il a une voiture.
（Il + a + une + voiture）
彼は車を持っている

2）ハイフン(－)または，-t-で切り離す

Avez-vous une voiture?
（Avez + vous + une + voiture）
あなたは車を持っているか？
A-t-il une voiture?
（A + il + une + voiture）
彼は車を持っているか？

3）アポストロフィ(')で切り離す

J'ai une voiture.
（Je + ai + une + voiture）
私は車を持っている

1)は，英語などヨーロッパ語でお馴染みの区切り方法です。というよりも，日本語·中国語など，単語の区切り記号がない言語のほうが少数派みたいです。

2)は主に，2つ以上の単語を連結する場合か，主語＋動詞の倒置を行った場合に用います。英語の倒置はことわりもなく行われ，紛らわしいですが，フランス語では，ちゃんと記号が入りますので，親切です。

例）Are you Japanese?
動詞　主語
Etez-vous Japonais?
動詞-主語
あなたは日本人ですか？

Suddenly cried he.
動詞　主語
Subitement crie-t-il.
動詞-t-主語
突然，彼は叫んだ。
(ただし，現在の「来来仏語」では，疑問文以外の倒置形は未対応です)

このハイフンを用いた倒置というのは，解析を進めるにあたって，有用な情報ですので，無視せずに，class TOKENのメンバ変数punctuationに記憶しておきます。

3)はエリジオン(Elision)といって，音便*1みたいなもので，母音が連続するのを防ぐ場合に使用されます。

例）le + arc en ciel → l'arc en ciel
ル・アルク・アン・シェル → ラルク・アン・シェル

というわけで，入力されたフランス語を単語で区切る場合は，"'"を単なる区切り記号としてではなく，すべての母音(a/e/i/o/u)を表す記号(ワイルドカード)として処理します。

実際のプログラムでは，ファイルf2j_main.cの関数，
AmatchS
区切り文字として，1)と2)を使用
AmatchSE
区切り文字として，1)～3)すべてを使用
の内部で処理しています。

*1　東京出身の人で，やってられない→，やってらんねえ,っていう人いますよね。最近はもう少し上品になって，やってらんなーい，かな？

単語の格・時勢・性

文字を単語ごとに区切ったら，その単語の意味を調べます。「来来仏語」では，約8000語の辞書データを持っていて，意味を調べるとは，各単語がどの辞書データと一致するのかを調べるということに相当します。

これは，単純に，名詞辞書・形容詞辞書・副詞辞書……と調べていきます。このとき，同じ綴りで異なる意味を持つものがあるので，すべての候補に関して，解析結果を保存します。

逆に，語源は同じでも，活用によって，綴りの異なる単語は，語源の辞書データを使用し，必要に応じて，活用形情報を参照できるような形で処理します。

たとえば，英語のbeには，

be	原形
am	1人称/単数/現在形
are	2人称/現在形
is	3人称/単数/現在形
being	現在分詞
been	過去分詞
was	1・3人称(単数)/過去形
were	2人称・複数/過去形

などの活用形がありますが，これらの活用形がそれぞれ独立の辞書を持っていては，不経済です。似たようなデータが，メモリ上にいくつもあるわけですし，開発の途中で"am"の辞書に変更を加えたら，"are"や"is"の辞書にも忘れずに同様の変更を加えなくてはいけません。特に，フランス語では動詞の活用が35種類もあるため，ぜひとも，活用語は原型語と同様の処理で行えるようにすべきです。
「来来仏語」では，辞書検索を行うと，

frPart	単語の品詞(表1参照) (どの辞書で見つけたか)
what	辞書へのポインタ
frAttrib	人称(格とか単数・複数)
frTense	時制情報

をclass TOKENの各メンバに格納します。

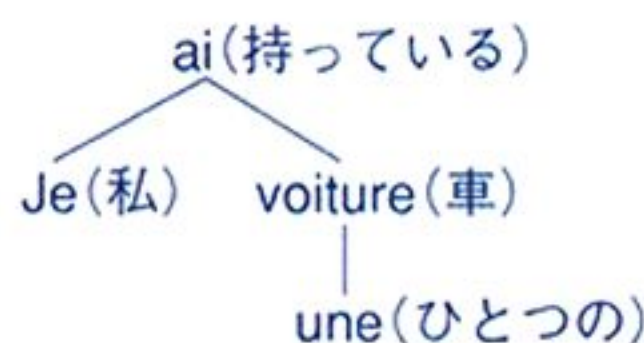
この情報は，日本語を表示する際の時制情報として役立つだけでなく(行く/行った/行くだろう...)，動詞の主語を特定する際にも役立ちます(2人称の動詞は2人称の名詞しか主語にとることができない)。

構文解析

単語の切り出しが終了したら，次は，構文の解析です。
最初，class TOKENは，

Je　ai　une　voiture.

のようにバラバラになっていますが，翻訳(フランス語の解析)が進むにつれて，ツリー構造に再構築され，

```
        ai(持っている)
       /        \
  Je(私)     voiture(車)
                 |
            une(ひとつの)
```

のように修飾・被修飾(あるいは，親と子)の関係付けが行われます。現バージョンでは，子TOKENの受け入れ方として，

subject　動詞の主語

図1　TOKENの連結例

表1　品詞の解析

品詞	プログラム中のラベル	Bit Pattern	例
冠詞	FR_PART_ARTICLE	0x00001000	une, des, la
名詞	FR_PART_NOUN	0x00002000	je, lui, voiture
動詞	FR_PART_VERB	0x00008000	chanter, voir
形容詞	FR_PART_ADJECTIVE	0x00010000	belle
前置詞	FR_PART_PREPOSIT	0x00020000	avec, devent
副詞	FR_PART_ADVERB	0x00040000	lentement,
疑問代名詞 疑問形容詞 疑問副詞	FR_PART_INTERROGATIVE	0x00080000	que, qui, où
関係代名詞 関係副詞	FR_PART_RELATIVE	0x01000000	que, qui, dont
間投詞	FR_PART_INDEPENDENCE	0x02000000,	oui, non
数詞	FR_PART_NUMETRIC	0x04000000	un, deux, trois
辞書データだけでは品詞のわからないもの	FR_PART_SPECIAL	0x08000000	aussi, de, des
接続詞	FR_PART_COMBINE	0x10000000	et, ou,

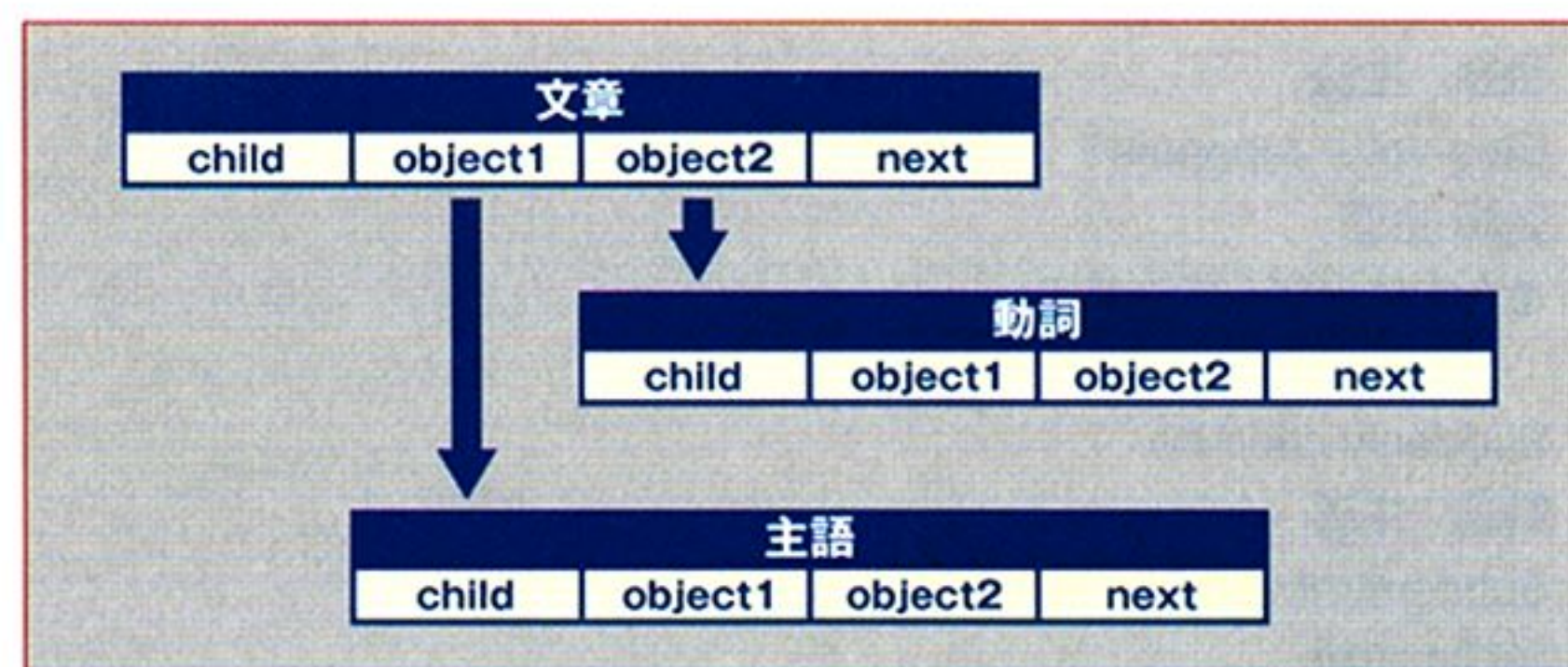
表2　構文解析の主なルーチン

CombineArticleNoun	冠詞と名詞の関連付け
CombineNounAdjective	形容詞と名詞の関連付け
CombineVerbAdverb	動詞と副詞の関連付け
CombineVerbObject	動詞と目的語の関連付け
CombineSubjectVerb	主語と述語(動詞)の関連付け
CombinePrepositioinObject	前置詞と目的語の関連付け
CombineParentPreposition	前置詞と親となる単語(名詞，動詞など)の関連付け

図2　以前のFR_PART_SENTENCE

child　　修飾するもの
object1　動詞・前置詞の目的語その1
object2　動詞の目的語2

があります。実際例は図1を参考にしてください。
構文解析そのものは，ファイルf2j_main.c中DoPass3から呼ばれているCombine～という関数で行われています(表2参照)。

私は弟とボールを投げた。

皆さんの中には，弟や妹を空に向かって投げつけた方は，どのくらいいるでしょうか？　では，続けて，ボールを投げた方は？
こういう質問をなんの前振りもなくできるのは，編集部の中でも私だけだ，と西川氏にいわれたことがあります。とりあえず，役割期待ということで質問してみましたが，常識からいうと，これは，

私は(ボールと弟)を投げた。

ではなく，

(弟と私)はボールを投げた。

の意味です。
日本語に限らず，自然言語は，人工言語(C言語などのプログラミング言語も人工言語の一種です)と違って，意味の曖昧さが残ってしまう場合があります。以下は，有名な例で，日本電子通信学会が開催されると，必ず誰かが引用する例です。

I saw the girl with telescope.
(J'ai regardé la fille avec une telescope.)

は，「with～」("avec")がどこに掛かるかによって，文章の意味が違ってきます。

```
F2J
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

C:¥WINDOWS>cd C:¥MYDOCU~1¥X2¥ISIGAMI¥F2J

C:¥My Documents¥x2¥isigami¥f2j>f2j -choice 5
仏> J'ai regard{ une fille avec un t{lescope dans le parc.
日> 望遠鏡といっしょで私は少女を(その)公園の中で見た。
日> 私は望遠鏡といっしょの少女を(その)公園の中で見た。
日> (その)公園の中の望遠鏡といっしょに、私は少女を見た。
日> (その)公園の中の望遠鏡といっしょの少女を、私は見た。
仏>
```

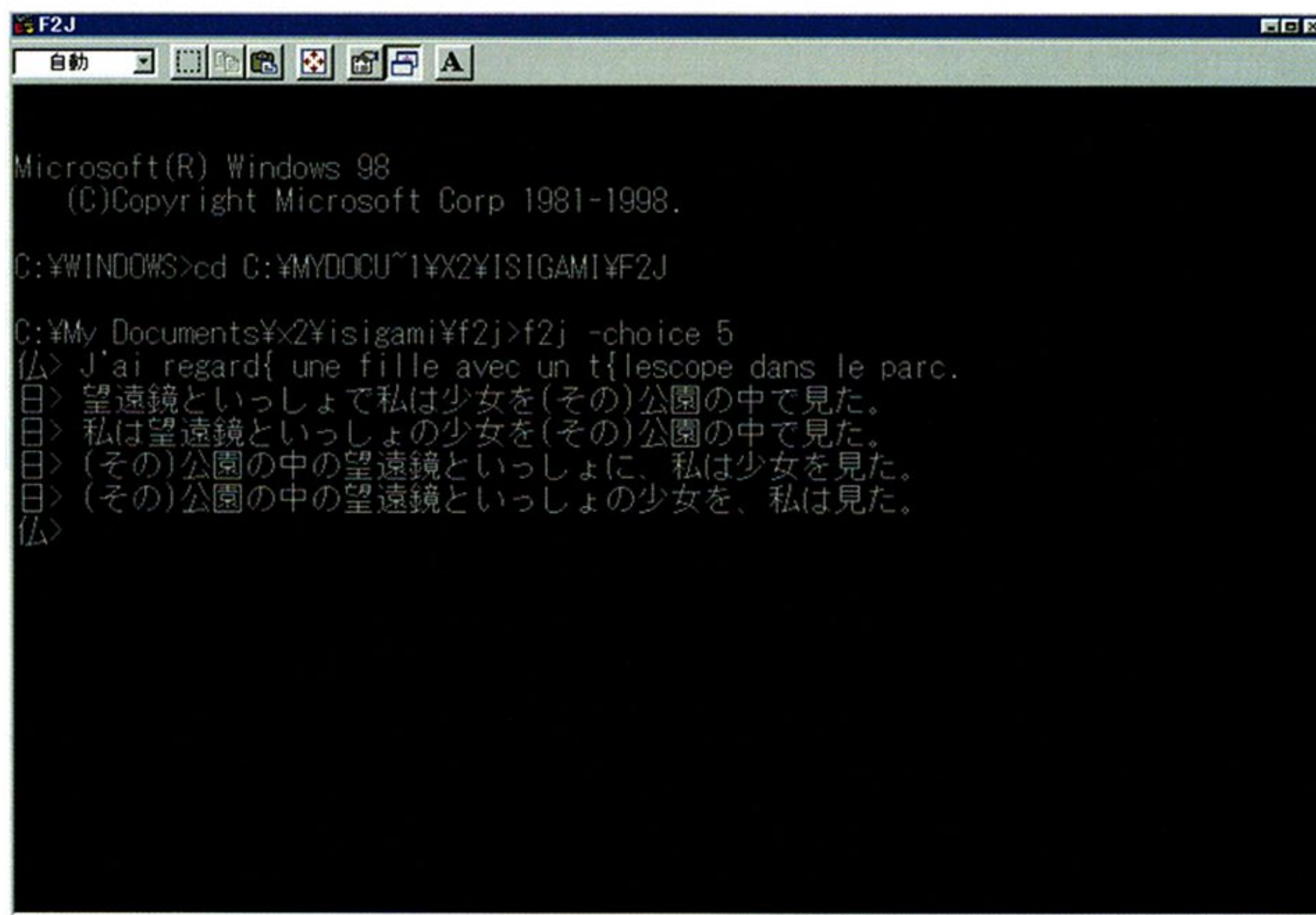



写真1

a) girlに掛かる場合

I saw (the girl with telescopre.)
望遠鏡を持った少女を私は見た

b) sawに掛かる場合

(I saw the girl) with telescope.
望遠鏡で私は少女を見た。

常識的に考えると，b)の訳ではないか，という気がしますが，この常識というのが曲者で，プログラム化するのは，大変です。
たとえば，

＜主語＞+ sea +＜人＞+ with +＜物＞
→＜主語＞は＜人＞を＜物＞で見る。

という変換規則を機械的に適用すると，

I saw the girl with blond hair.
(J'ai regardé la fille avec le cheveu blond.)

a) girlに掛かる場合

I saw (the girl with blond hair) .
金髪の少女を私は見た

b) sawに掛かる場合

(I saw the girl) with blond hair.
金髪で私は少女を見た。

のうちで，常にb)が適用され，どうやってツッコミを入れればよいのか，まったくわからない文章になってしまいます。
これは，面白いトピックスではあるのですが，今回のバージョンでは，単に，複数の解釈が考えられる場合は，(どんなに情けないものでも)プログラムで判断せずに，可能性のある訳をすべて出力して，ユーザーに選んでもらおうというスタンスでいきます(写真1)。
候補結果を複数出力し，最終的にユーザーに最適解を選んでもらうということは，解析中に曖昧な部分が出てきたら，候補1，候補2，……，候補n，のすべてをメモリ上に展開し，後続の処理を行うということです。これは，図3のように行っています。
この別の候補を作り出す関数が，CopyCurrentTreeで，返り値として，現在注目している候補nの中のTokenが，候補n+1ではどこにあるかというポインタを返します。つまり，この関数実行後，

曖昧さの残るTokenの候補n中の位置
曖昧さの残るTokenの候補n+1中の位置

が得られるわけで，それぞれに別な処理を行い。それぞれの候補に対し，後続の処理を行います。このとき，候補nと候補n+1にまったく同じ処理を行っていたのでは，候補n+1は，まったく同じ条件で候補n+2を作ってしまい，無限ループに入ってしまいます。これを避けるため，候補nと候補n+1を区別する情報が必要です。このため，Tokenの中には，

frPartParent：親になる可能性のあるTokenの種類のビットマスク
frPartChoice：自分がなる可能性のあるTokenの種類のビットマスク

という2つのメンバ変数が用意されています。このビットマスク情報を見ながら，同じ候補を考えないようにします(ビットマスクの値は表I参照)。先ほどの例でいうと，「avec～」(with)がどこに掛かるか曖昧だったので，

1）検討する前，

frPartParent=
FR_PART_ALL (0xffff0000)

2-a) fille (girl)に掛かる，とみなす候補

frPartParent=
FR_PART_NOUN (0x00002000)

2-b) regarder(saw)に掛かる，とみなす候補

frPartParent
=FR_PART_VERB (0x00008000)

になります。

root
subject | child | object1 | object2 | next
単語1
subject | child | object1 | object2 | next
候補1
subject | child | object1 | object2 | next
単語1
subject | child | object1 | object2 | next
候補2
subject | child | object1 | object2 | next
単語1
subject | child | object1 | object2 | next

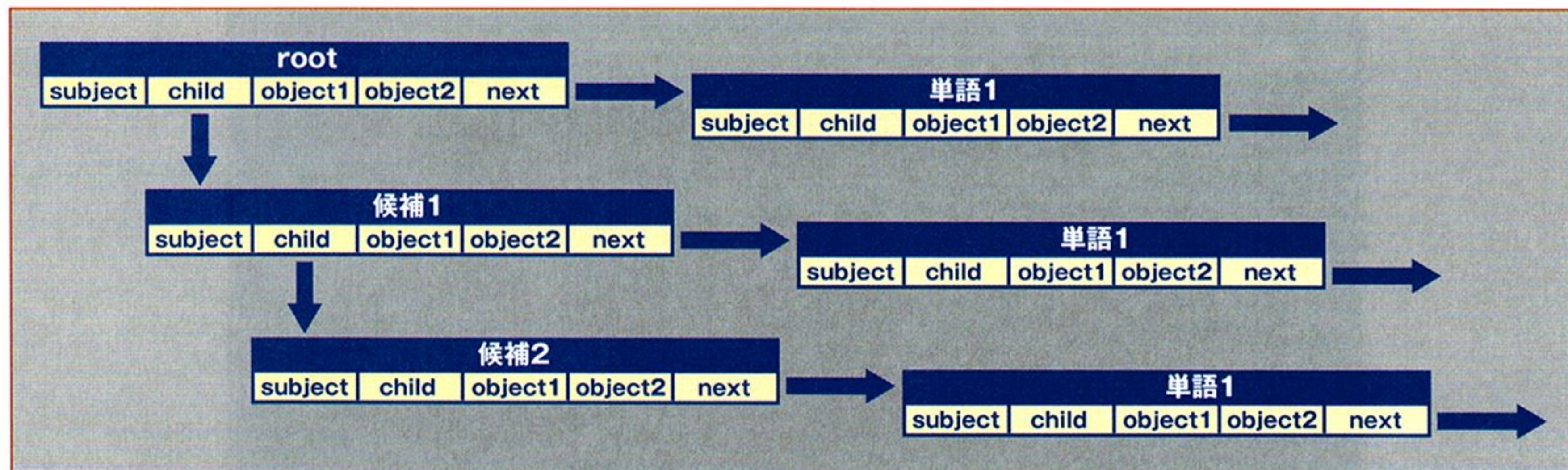



図3　複数の候補

従来比速度3倍

復刊1号に収録したVersion0.30から，本号に収録したVersion0.40で，品詞の種類をひとつ減らしました。

以前は，主語＋述語の組み合わせを図2のように“文章”(member frPart == FR_PART_SENTENCE)という品詞として，扱っていましたが，これを図1のように，TOKENにsubjectというメンバを加え，“動詞”(member frPart == FR_PART_VERB)という品詞の範囲内で扱うようにしました。

このことにより，

動詞 ＋ 副詞 → 動詞句
文章 ＋ 副詞 → 文章

という処理を統一して行えるようになりました。

さらに，先ほど例にとった前置詞なら，

1）名詞に連結する前置詞(girl with telescope)
2）動詞に連結する前置詞(saw with telescope)
3）文章に連結する前置詞((I saw the girl)with telescope)

という3種類の可能性のうち，3)の候補を扱う必要がなくなり，処理量が，1/3軽減されました。この前置詞ひとつにつき，1/3軽減というのがミソで，

J'ai regardé une fille avec telescope dans le parc.
(I saw the girl with telescope in the park.)

というように2つの前置詞があると，

・“文章”ありの場合
3(avecの候補)×3(dansの候補)=9

・“文章”なしの場合
2(avecの候補)×2(dansの候補)=4

となって，処理する組み合わせが減るわけですから，2倍以上処理量が軽くなります。ここで，処理量というのは，計算時間と，メモリ消費量の両方です。

さらに，フランス語最頻出語である“de”というのは，前置詞のほかにも冠詞(不定冠詞/否定冠詞/部分冠詞形容詞をともなう冠詞)にも使いますから*2，

例)Au diner, nous mangeons du potage chaud, des petits poissons, de la viande, de la salade, du fromage blanc, des fruits de saison et nous buvons du vin rouge.
(夕食に，私達は，暖かいポタージュ，小魚，肉，サラダ，ホワイトチーズ，季節の果物を食べ，赤ワインを飲む。*3)

なんていうのは，

・“文章”ありの場合4^8 = 65536
・“文章”なしの場合3^8 = 6561

と10倍近い効果があります。実際には，これにカンマ(“,”)の意味を複数考慮しなければいけないので，後者の場合でも候補数は5000を超えてしまい，現バージョンの「来来仏語」では，残念ながら処理しきれません。

指数関数的に増えていく処理量関数の基数を減らすべきなのは，落ち着いて考えれば，当たり前のことなのですが，当初なかなか気づかず，ずいぶん回り道をしてしまいました(なにせ，1号と2号の間，この改造につきっきりでしたから)。

*2「電車でGo！」とか「読んde!! ココ」というような手段や対象物を表す使い方は，仏語にはありません。「パフィーdeルンバ」は，私にはわかりません。

*3 “du”，“des”というのは“de”の活用形の一種です(縮訳)。ちなみに，下線の“de”が部分冠詞，それ以外が前置詞(季節の果物)です。

参考文献
1)大場五夫「パソコン翻訳入門<翻訳ツールの上手な使い方>」日本理工出版会　1995
2)倉田清「仏文和訳の実際」大修館書店 1995

先月号の訂正

強行スケジュールのため，Oh!X復活1号206ページの写真では，正しくないフランス語が入力されています。入力がでたらめなので，出力がでたらめなのは当然なのですが，キャプションに“翻訳の実行結果。うーむ……”とか入っているので「来来仏語」の性能に必要以上に疑問を持たれるといけないので，少し弁解をします。

1行目　：単純なスペルミス。3行目が正解
7行目　：不明
11行目：不明
17行目：不明

未来仏語の課題

「来来仏語」は，私が大学1年生のときに使った教科書をひととおり実装し終え，現在，参考文献1を片手に，各種アルゴリズムの補強，パラメータの調整を行っています。この本は，基礎編，応用編Ⅰ，応用編Ⅱの3章から構成されており，基礎編の例文は，翻訳規則ごとにまとめられていて，通し番号がふってあるところが気に入っています。〝フランス語の基本的な短文から上級程度の論説文まで，……〟(まえがき)を対象としているので，すべてを自動翻訳で扱うには無理があるのですが，基礎編(25章，641例文)のひととおりの例文を当面の目標としてしています。現在，例文400あたりまで，対応できるようになりましたが，残念ながら，以下の文章は扱えませんでした(今回は例文1～200です)。

〝熟語未サポートのため〟，と書いてあるのは，やればできるが，まだ，熟語をどのように(統一的に)扱うかを決めていないので，とりあえず無視している個所です。ひと口に熟語といっても，動詞＋形容詞＋名詞や名詞＋前置詞＋名詞，などさまざまな組み合わせの上に成り立っているのがわかると思います。いまでも，やろうと思えば対応できますが，熟語の扱いを正式に決めたあとでもう一度やり直すことになるので，対応していません。

それ以外は，降参している個所です。本来，胸を張って紹介するものでもないのですが，読者の方やほかのライターの方が助け舟を出してくれることを，切に願って，以下にリストアップします。専門家から見れば，笑ってしまうかもしれませんが，自動翻訳の右も左もわからず始めたプログラムですので，どんなアドバイスでも大歓迎です。

1．不定冠詞・定冠詞・指示形容詞

25 熟語tour à tour(ときどき)が未サポートなため。

2．être，avoir・所有形容詞

32 Il est grand et maigre, ses cheveux sont bruns, ses yeux gris.
(He is tall and thin, his hair is brown, his eyes gray.　彼は背が高く，痩せていて，髪は茶色，目は灰色だ)

yeuxとgrisの間にêtre動詞が省略されているので，名詞節として認識してしまう(目は灰色だ→灰色の目)。

45 Je pense, donc je suis.(Descartes)
(I think, so I am. 私は考えるので，私はいる。→われ思う。故にわれあり)

suis(原型はêtre。英語のbe動詞に相当。仏語でもっとも頻繁に使用される動詞のひとつ)を自動詞として認めると，ほかの文の解釈で，候補数が莫大になるので，実装していない。もう少し，プログラムの形が整ってから，再考する。

3．部分冠詞

54 本文参照。

65 Pas de nouvelles, bonnes nouvelles.(Poverbe)
(No news, good news. 便りのないのがよい知らせ)

動詞がないので，困っています。

4．疑問形容詞・感嘆文

いまのところ，問題なし。

5．命令法

89 熟語de mainière à(～できるように)未サポートのため。

92 Entrez par la porte étroite, car la porte large et le chemin spacieux menent à la predition.
(Enter through the small door, because the big and wide door leads to a ruin.　狭い門から入りなさい。なぜなら，大きくて広い門は滅びに通じているから。聖マテオの福音)

接続詞car(なぜなら)，定冠詞la(その)，名詞porte(門)

→　なぜなら，その門は，～

を，

名詞car(バス)，人称代名詞la(それを)，動詞porte(運ぶ)

→　バスはそれを運び，～

と取り違えてしまう。「来来仏語」では，〝文章〟(＝主語＋述語)の検出を高く評価するので，後者(狭い門から入りなさい。バスはそれを運び，広い門は滅びに通じているから。3つの文章)が大きな評価値を得てしまう。冠詞がない一般名詞を大きく減点する手もあるが，実際には，教科書ほど厳密な規則がないようなので，かえって「来来仏語」の範囲を狭めてしまいそう(例)arc en cielはcielが無冠詞。同じ名詞＋前置詞＋名詞でも，cafe au(= a le) lait はlaitに冠詞がつく)。

6．非人称動詞

110 Pour tout peindre, il faut tout sentir.(Lamaritine)
(To paint everything, it is neccesarry to feel everything.　すべてを描くためには，すべてを感じることが必要だ)

目的語(tout)と動詞(peindreとsentir)が倒置している。

113．散文詩のため，省略。

7．比較

149. La raison du plus fort est toujours la meilleure.(La Fontaine)
(The reason of the strongest(person) is always best.　もっとも強い(者)の理屈は常にもっともよい)

fort(強い)の後ろにhomme(者)が省略されている。用途が不明な形容詞は，無条件に名詞化しても良いかを検討中。

154. 熟語vaut mieux que(する価値がある)未サポートのため。

155 La passion fait souvent un fou du plus habile homme, et rend souvent les plus sots habiles.(La Rochefoucauld)
(The passion sometimes makes the greatest person stupid, and changes the most stupid person greatest(person).　情熱はもっとも有能な者をしばしば愚かにし，もっとも愚かな者をしばしば有能にする)

habiles(有能な)の前にhomme(者)が省略されている。149と同様。

8．助辞による時

いまのところ，問題なし。

9．複合過去

167 熟語tout à l'heure.(いましがた)未サポートのため。

168 熟語passer une nuit blanche(白い夜を過ごす→徹夜する)未サポートのため。

174 La chair est triste, hélas! et j'ai lu tous les livres.(Mallarme)

文書の真ん中に特殊記号(！)が入っているので。

10．単純未来・前未来

187 Vous aurez mal compris.
(You might misunderstand.　あなたは誤解したようだ)

直説法前未来には，未来の一時期に完了する動作/婉曲表現の2種類の意味があるが，「来来仏語」は前者としてすべて翻訳してしまう(あなたは誤解するだろう)。

188 Il aura encore fait des bêtises.
(He might play the fool again.　彼はまた馬鹿なことをしたらしい)

187と同じ理由(彼はまた馬鹿なことをするだろう)。

11．受動態

いまのところ，問題なし。

D3D IMを使う

菊地 功 Kikuchi Isawo

Direct3Dを使ううえでの山となるのが，Immediate Modeへの移行だろう。データ構造などはほとんど用意されていないので書いていかなければならないものの，Retained Modeよりもはるかに柔軟な処理ができる。ここではDirectX6から追加されたフィーチャであるバンプマッピングを行ってみる。

Direct3Dを使うには2つの道がある。ひとつがこれまでやってきたDirect3D Retained Mode(以下，RM)，もうひとつがより低レベルな処理を行うDirect3D Immediate Mode(以下，IM)だ。RMのほうの記事では「RMはこれだから……」とか「IMだったら……」みたいな雰囲気が漂っていたと思う。勘のよい読者ならばおわかりかもしれないが，そう，第2回めにして早くも私は融通の利かないRMにはうんざりしてしまったのだ。そんなわけで，IMにも手を出そうと思う。以前は編集の(U)氏に「やれー」といわれても，「やだー」と逃げ回っていたのだが，やはり避けては通れない道らしい。ただ，IMをいじるのは初めてなので，思いのほか厄介だったらRMにとんぼ返りするかもしれない。あ，IMが順調にいっても，RMはしばらく並行するつもりだからRMでやってる人も大丈夫。

RMのところでも述べたように，IMを使えばDirectX6の新機能であるマルチテクスチャだろうがバンプだろうがステンシルバッファだろうがやりたい放題だ(能力が追いつけばの話)。筆者としては，いちばん興味があるところはバンプマッピングであるので，今回は最終的にそれを目指してみたいと思う。

男は黙ってイミディエイト

ところで「IMを使う」といっても，方法は2つある。

1) RMを拡張する形でIMを使う

RMにはユーザービジュアルというものがある。これは，ユーザー(プログラマ)が「目に見えるもの」として定義し，フレームにくっつけておけるものだ。ユーザービジュアルの表示の必要があるときには，コールバック関数が呼ばれるようになっている。そこでIMを使ってプリミティブをレンダリングしてやれば，一応「IMを使った」ということになろう。RMはIMの上に乗っかっているので，初期化部分もRMが勝手にやってくれる。

2) ストイックにIMだけで頑張る

IMはフレームの概念などがないので，その辺りは自分でクラスなどを作って管理することになる。当然RMでは自動的にやってくれた変換なども，自前でやらなければならない。数学の教科書を見ると眠くなってしまう人や，数字の羅列に遭遇するとパニクってしまう人などには，あまりおすすめできない。4×4列で行進してくる数字に襲われる悪夢でノイローゼにはなりたくないだろう。

とはいえ，本記事は2)のほうでいくことにする。所詮1)はその場しのぎであり，結局RMの呪縛に縛られてIMの柔軟さが失われてしまうことが目に見えているからだ。そもそも2)のほうが圧倒的に美しい。

というわけで，基本を少々。

Direct3Dは，配置したプリミティブをレンダリングするのに，3つの変換を必要とする。まずはワールド変換。RM的にいえば，フレームの座標を絶対座標に直す変換であり，フレームそのものであるといってもよい。子フレームを作ろうと思ったら，親と子を合成した変換が，そのフレームに載っているプリミティブに対するワールド変換となる。

2つめはビュー変換。ワールド空間に配置されたプリミティブやライトを，視点の座標に直す変換だ。RMのカメラフレームに相当する。最後は射影変換。これは2つの変換とは性格が異なり，パースペクティブ法の要となる変換だ。これは，カメラの視野とフロントクリップ，バッククリップで囲まれた視錐台を，立方体(直方体)に変換する(図1 投影変換)。この3つの変換を頂点に対して順に行い，最終的に得られたxy座標がディスプレイにスケーリングされると，画面上でのxy座標となる。ちなみにZバッファとは変換後のzを使用する手法であり，変換前のzを使うのがwバッファである。

これらの変換は，4×4のマトリクスで表すことができる。

$$[\,x'\ y'\ z'\ 1\,]=[\,x\ y\ z\ 1\,]\begin{bmatrix} M11 & M12 & M13 & M14 \\ M21 & M22 & M23 & M24 \\ M31 & M32 & M33 & M34 \\ M41 & M42 & M43 & M44 \end{bmatrix}$$

これは，次の座標に展開される。

$x' = x \times M11 + y \times M21 + z \times M31 + 1 \times M41$
$y' = x \times M12 + y \times M22 + z \times M32 + 1 \times M42$
$z' = x \times M13 + y \times M23 + z \times M33 + 1 \times M43$

一般的な変換としては，移動，回転，スケーリングが挙げられる。

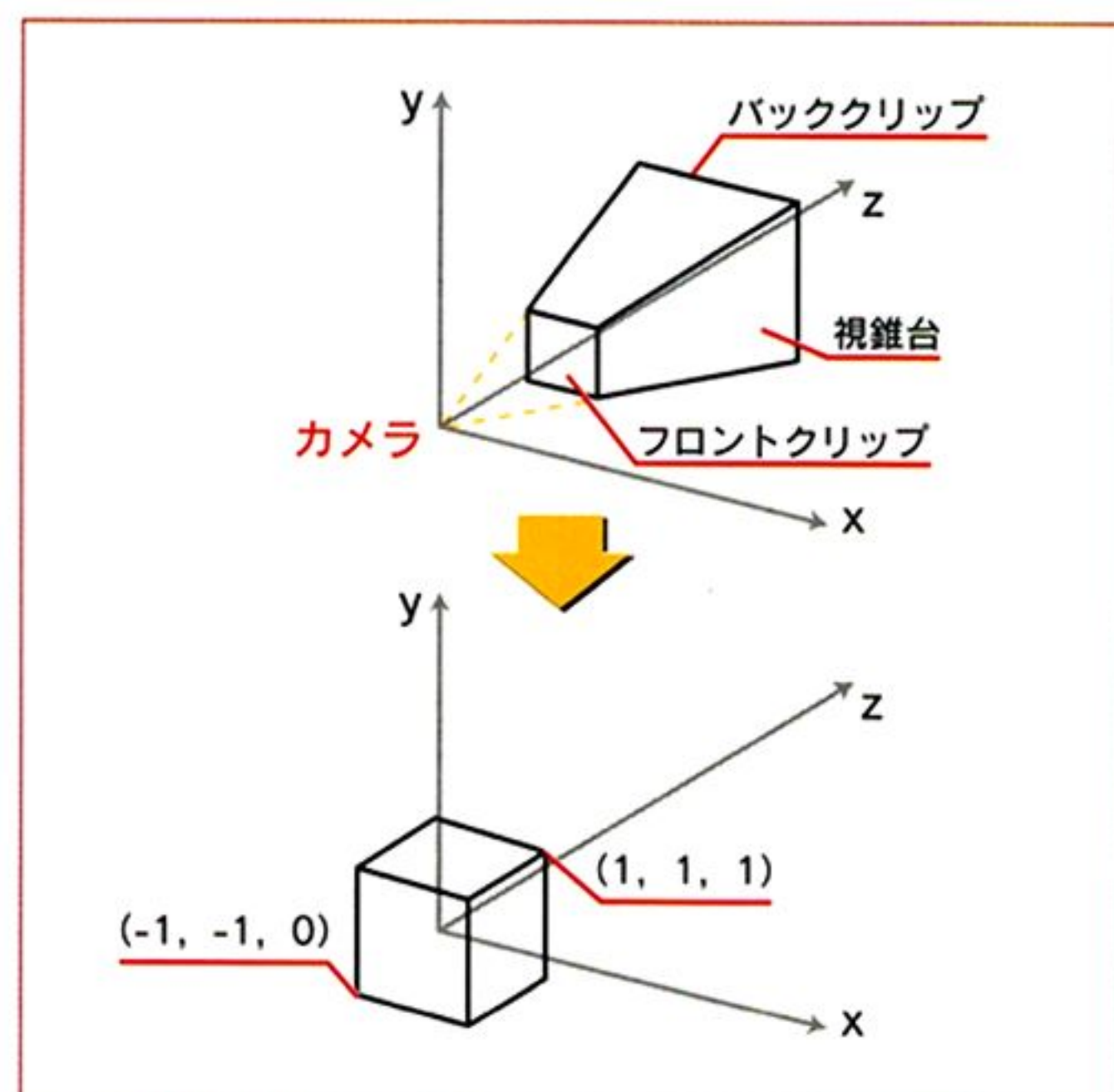
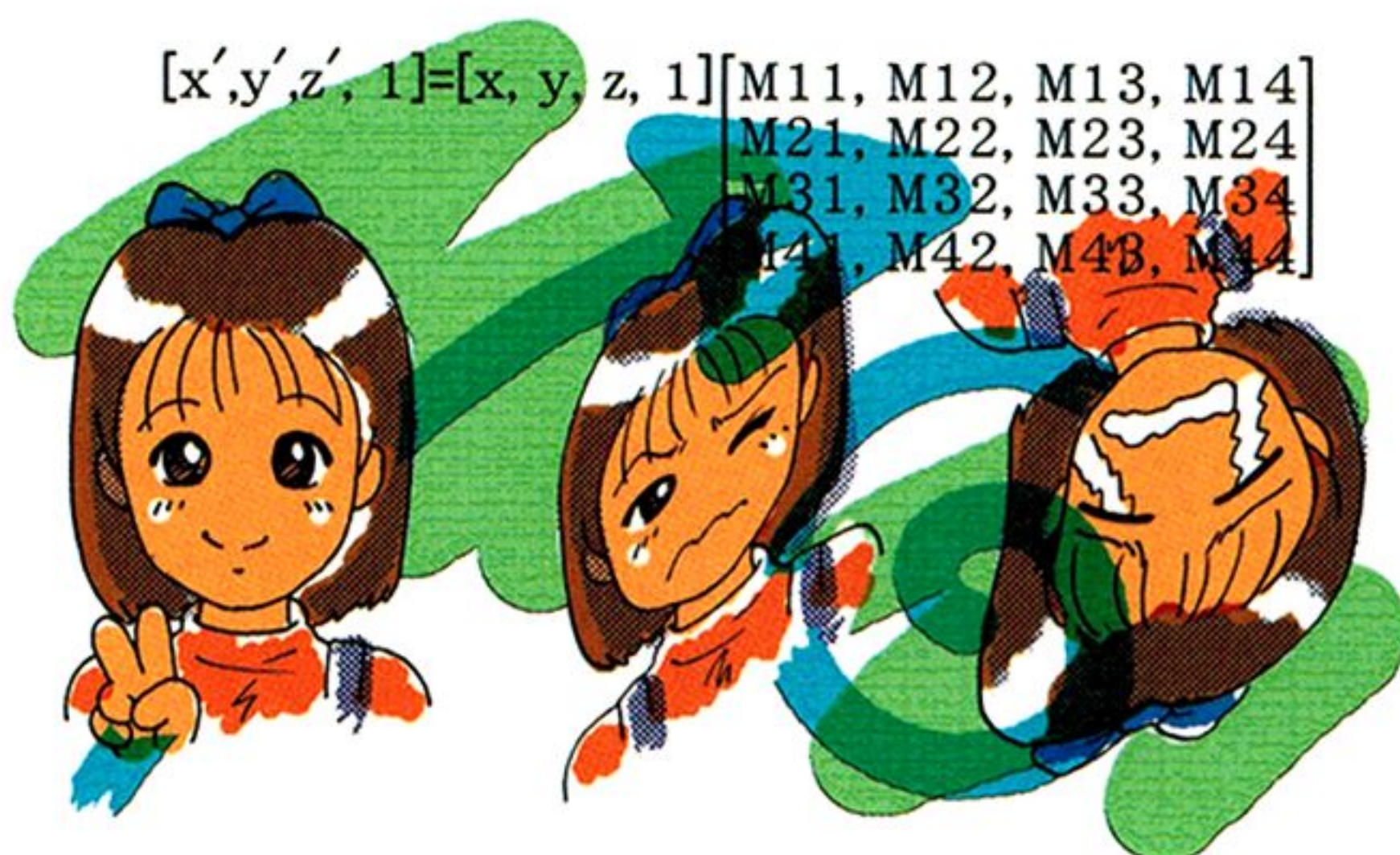



図1 投影変換

・移動

$$[x'\ y'\ z'\ 1]=[x\ y\ z\ 1]\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\T1&T2&T3&T4\end{bmatrix}$$

$x'=x+T1$
$y'=y+T2$
$z'=z+T3$

・回転（例：z軸周り）

$$[x'\ y'\ z'\ 1]=[x\ y\ z\ 1]\begin{bmatrix}\cos\theta&\sin\theta&0&0\\-\sin\theta&\cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$x'=x\cos\theta-y\sin\theta$
$y'=x\sin\theta+y\cos\theta$
$z'=z$

・スケーリング

$$[x'\ y'\ z'\ 1]=[x\ y\ z\ 1]\begin{bmatrix}Sx&0&0&0\\0&Sy&0&0\\0&0&Sz&0\\0&0&0&1\end{bmatrix}$$

$x'=x\times Sx$
$y'=y\times Sy$
$z'=z\times Sz$

ワールド変換とビュー変換は，これらのマトリクスの合成で与えることになるが，実際はいちいちマトリクスを設定しなくても，SDKに付属するユーティリティ関数群を利用すればいいので，あまり構える必要はないだろう。

なお，IMではレンダリング方法として，実行バッファとドロープリミティブという方法が用意されている。実行バッファというのは，その名のとおりバッファを作り，そこに命令をどかどかと書き込み，ほいっとデバイスに渡してやる方法だ。いわゆるバッチ実行である。

しかし，それではバッファを作るのが面倒だということで，DirectX5からはドロープリミティブが使えるようになった。これは，早い話が実行バッファに描き込む命令1つひとつを，関数を使って単独でコールできるようにしたものだ。

基本的にできることはどちらも変わらない。実行バッファはCPUとビデオ側にそれぞれ遊び時間ができてしまうから無駄が多いとか，いやいやその間にCPUは当たり判定とかほかの処理ができるから有利だとか，どの道ジオメトリ演算は(いまは)CPUでやってるんだから関係ないとか，いろいろな話があるが，その辺はSDKのヘルプに結論が出ている。「どっちでもたいして変わらないから，使い慣れたほうを使うべし」ということで，本記事ではドロープリミティブを使うことにする。

ポリゴンを描く

いきなり「おりゃ〜バンプだ〜」といって突っ走ろうとしても，すぐに迷子になってしまうのは目に見えているので(実際迷子になった)，まずは軽くポリゴンを表示してみよう。RMで使ったEnumDeviceやサンプルはスケルトンとして利用できる。Direct3Dオブジェクトやデバイスの作成まではほぼ同じだ。

ただし，最新のIDirect3D3とIDirect3DDevice3のオブジェクトを作ること。ここで注意が必要なのだが，IDirect3D3からはRampドライバがサポートされなくなってしまった。SDKのヘルプによると「Rampドライバはあまり使われていないから」ということなのだが……別になくすことないんじゃないかとも思う。いままで「別に一3Dなんていらねーよ，とりあえずエミュレーションでも動くし」とかいって，古いビデオカードで頑張ってた人は，そろそろ考える時期なのかもしれない。RGBドライバやMMXドライバは問題なく従来どおり使える(ただし，MMXドライバはIDirect3D3::EnumDevices ()では列挙されないため，列挙したければIDirect3D2::EnumDevices ()を使うこと)。

RM用の初期化関係はさっくりとご退場いただくとして，いままでRMにやってもらっていた初期化部分をIMで記述していこう(リストPlane.cpp抜粋)。まずはビューポートをIDirect3D3::CreateViewport ()で作って，それをIDirect3DDevice3::AddViewport ()でデバイスに追加する。特になんということもないので説明は省略する。次にビューポートにクリッピングボリュームをセットする。これにはD3DVIEWPORT2構造体を使用する。

```
struct D3DVIEWPORT2 {
    DWORD        dwSize;
    DWORD        dwX;
    DWORD        dwY;
    DWORD        dwWidth;
    DWORD        dwHeight;
    D3DVALUE     dvClipX;
    D3DVALUE     dvClipY;
    D3DVALUE     dvClipWidth;
    D3DVALUE     dvClipHeight;
    D3DVALUE     dvMinZ;
    D3DVALUE     dvMaxZ;
};
```

dwSizeは例によって，この構造体のサイズを入れる。次の4つはビューポートに対応させるディスプレイピクセルの設定で，画面全体に表示したいのならdwXとdwYには左上隅を表す0を，dwWidthとdwHeightにはそれぞれ画面の幅と高さを指定する。

さてそのあとだ。ここからは先ほどの図のうち，射影変換後の直方体の設定をする。dvClipX，dvClipYは左上となる座標，dvClipWidth，dvClipHeightはそれぞれ幅と高さだ。ここで，範囲をディスプレイに表示させるアスペクト比にあわせないと，つぶれてレンダリングされてしまう。画面の縦÷横をaspectとすると，dvClipXに−1.0，dvClipWidthに2.0を入れ，dvClipYにaspect，dvClipHeightに2×aspectを入れるのが一般的らしい。

最後のdvMinZとdvMaxZはZ方向の範囲で，普通0.0と1.0を指定する。こうしてできたD3DVIEWPORT2構造体を，IDirect3DViewport3::SetViewport2 ()に渡し，そのビューポートをIDirect3DDevice3::SetCurrentViewport ()でカレントにしてやればセット完了だ。

では，シーンの作成に移ろう。まずは射影変換を行う投影マトリクスを作る。これには，SDKのサンプルに付属のD3DUtil.cppとD3DUtil.hを使うと簡単だ。デフォルトでは，C:¥mssdk¥samples¥Multimedia¥D3DIM¥src¥D3DFrameおよびC:¥mssdk¥samples¥Multimedia¥D3DIM¥includeに入っている。または，cppをライブラリにしたd3dframe.libがC:¥mssdk¥samples¥Multimedia¥D3DIM¥libにあるので，それをリンクしてもよい。このなかで，投影マトリクスを作るD3DUtil_SetProjectionMatrix ()という，そのまんまの関数が用意されている。

```
HRESULT D3DUtil_SetProjectionMatrix (
    D3DMATRIX& mat,
    FLOAT fFOV,
    FLOAT fAspect,
    FLOAT fNearPlane,
    FLOAT fFarPlane
);
```

それぞれの変数の意味は，図2のとおりだ。ちなみにRM(つまりIDirect3DRMViewport2)のデフォルトはfFOV，fAspect，fNearPlaneが1.0，fFarPlaneが100.0である。これで作成したマトリクスを，IDirect3DDevice3::SetTransform ()に第1引数D3DTRANSFORMSTATE_PROJECTIONとともに渡してやればOKだ。今度はビューマトリクス。これもD3DUtilに関数が用意されている。

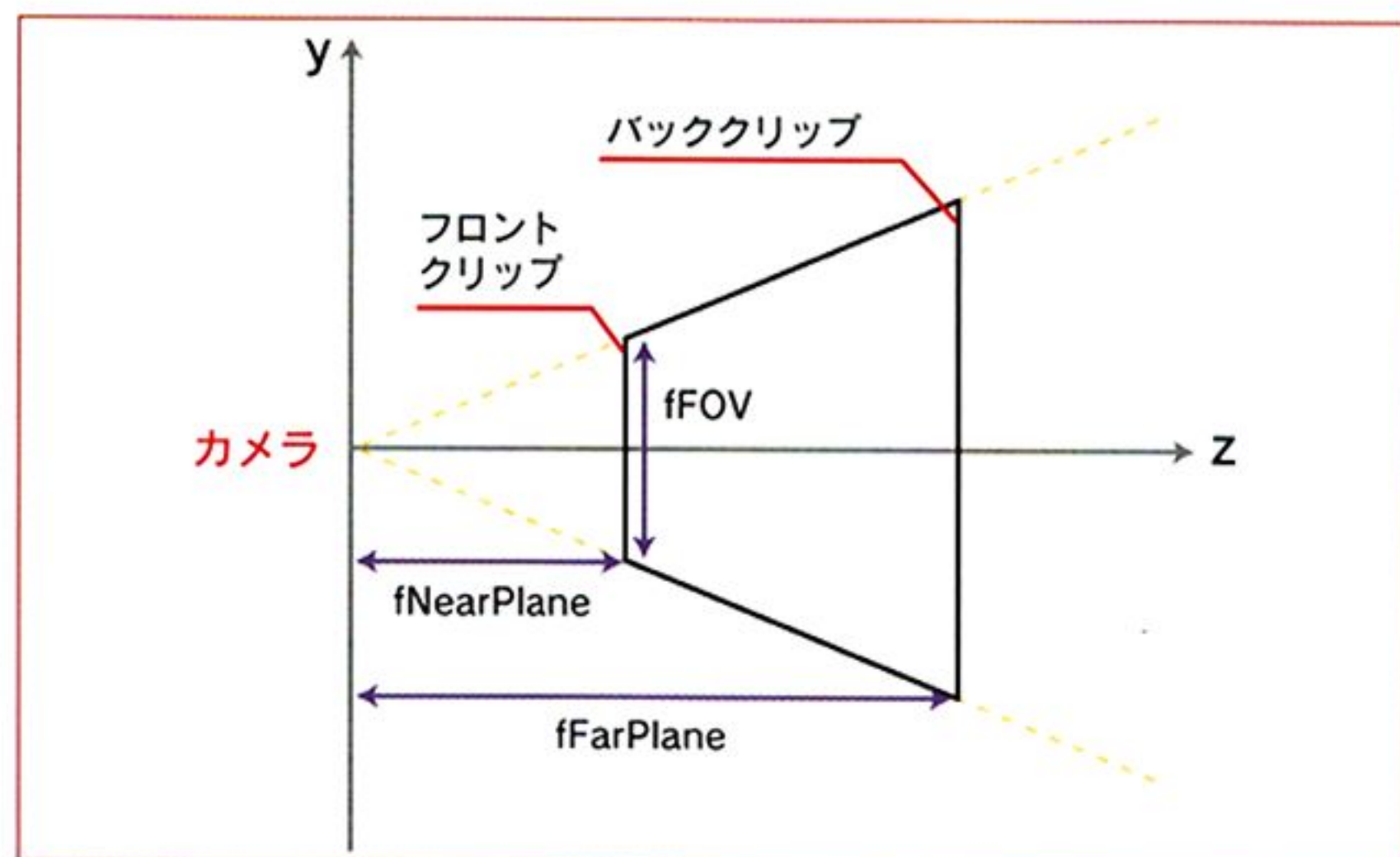



図2　投影マトリクスのパラメータ

```
HRESULT D3DUtil_SetViewMatrix (
  D3DMATRIX& mat,
  D3DVECTOR& vFrom,
  D3DVECTOR& vAt,
  D3DVECTOR& vWorldUp
);
```

vFromはワールド空間における視点の位置，vAtは視線の先，vWorldUpはビューポートの上向きを示すベクトルである。今度はこれをD3DTRANSFORMSTATE_VIEWとして，IDirect3DDevice3::SetTransform ()に渡す。今回は投影マトリクスもビューマトリクスも渡したらそのままであるが，もしカメラを動かしたければ，ビューマトリクスを逐一作成してIDirect3DDevice3::SetTransform ()に渡すことになる。

次にマテリアルとライトを作ろう。この辺はRMにちょっと似ている。この2つを初期化するのも，D3DUtilにある関数で簡単にできる。詳しくは述べないが，マテリアルは色を，ライトはライトの種類と座標を適当に放り込めば，とりあえず使えるオブジェクトができる。できたライトはビューポートに追加し，マテリアルはIDirect3DDevice::SetLightState ()にD3DLIGHTSTATE_MATERIALとともに渡す。

おや？　RMではマテリアルはメッシュにセットするのに，なんでデバイスがここで出てくるの？　しかもライトステート？　う～ん，その謎はあとで解明することにしよう。

そのあとは適当にステートを設定して，クリップステータスとワールド変換マトリクスを初期化する。クリップステータスっていうのは，レンダリングされた領域を保存しておく構造体。以前描いたフレームを消すのに，全画面初期化するよりは，この情報を使って描かれた領域だけを塗りつぶすほうが高速だ。フルスクリーンで実行された場合は，プライマリとバックバッファで画面が2画面あるので，クリップステータスも2つ用意してある。今回はなんとなく時間管理をしてみようと思ったので，GetTickCount ()で時間を取得して，終了。あれ，これで終わり？　メッシュみたいなものは作らなくていいの？　うん，それもあとでね。

いよいよレンダリングである。IMではレンダリングの最初でIDirect3DDevice3::BeginScene ()を，最後でIDirect3DDevice3::EndScene ()を呼ぶことになっている。まあ，その辺はそういうもんだと思っておけばいい。

まずは先ほどのクリップステータスでバッファをクリアする。IDirect3DViewport3::Clear2 ()でバックバッファとZバッファを同時にクリアできるのだが，ここでちょっと注意が必要。ウィンドウモードの場合はそもそもフリップで画面を切り換えるわけではないので，直前に取得したクリップステータスでバックバッファ，Zバッファともにクリアすればよい(つまりクリップステータスはひとつしか使わない)。

フルスクリーンモードの場合，プライマリとバックバッファが交互にレンダリングされ，そのためにクリップステータスを2つ作ってあるのはすでに述べた。だから，バックバッファはひとつ前のクリップステータスを使ってクリアする。しかし，Zバッファはひとつである。したがって，こちらがクリアに使うクリップステータスは直前のものであり，よって，バックバッファとZバッファは別々にクリアしなければならない。

そろそろ山場だ。まず，デバイスのクリップステータスを初期化しておく。このクリップステータス外を描かない，という意味ではなく，更新領域を空にしておく，とでもいえばいいだろうか。仮に現在クリップステータスの領域が(x1，y1)－(x2，y2)だったとして，そこに(x，y)座標にプリミティブが描かれると，

```
if ( x<x1 )  x1 = x;
if ( x>x2 )  x2 = x;
if ( y<y1 )  y1 = y;
if ( y>y2 )  y2 = y;
```

といったコードで表せる処理が自動的に行われ，領域がどんどん広がっていく。したがって，なにかが描画されたときにクリップステータスに確実に反映させるためには，x1とy1に取りうる最大値，x2とy2には最小値を初期値として設定しておけばよい。

適当に時間管理して回転角を決めたら，D3DUtilのD3DUtil_SetRotateXMatrix ()を使って，X軸周りにその角度だけ回転するワールドマトリクスを設定する。とどめはIDirect3DDevice3::DrawPrimitive ()でレンダリング。

って，おい，ポリゴンデータはどうなったんだよう。ちゃんと渡してるじゃん，DrawPrimitive ()の第3引数で。そう，実はIMでは，ポリゴンは生の配列であって，それを扱うインタフェイスは特に用意されていないのだ。でもって，DrawPrimitive ()が呼ばれたとき，引数として呼ばれたポリゴンを，そのときに設定されているライトとレンダーステートでレンダリングする。さっきマテリアルをライトステートに設定しただろう。つまりこのポリゴンはさっき設定したマテリアルでレンダリングされ，もし異なる色のポリゴンを描こうと思ったら，マテリアルを設定し直して，再びDrawPrimitive ()することになる。

なんかものすごい回りくどいことをしているような気もするが，もともとが実行バッファでバッチ実行するためのオペレーションをばらしてしまったんで，こうなってしまうんだろう。

ではなぜ「ライト」ステートなのか。Direct3Dのレンダリングパイプラインは3つのモジュールからなっている。変換モジュール，照明モジュール，ラスタ処理モジュールだ。変換モジュールとは，ワールド変換マトリクス，ビュー変換マトリクス，投影マトリクスを使って，頂点を投影空間に変換するモジュールである。照明モジュールは，ライトやプリミティブの法線などから頂点の色を計算する。ラスタ処理モジュールは，それらの演算結果から，フレームバッファにレンダリングを行う。このなかで考えれば，「プリミティブの色」は明らかに照明モジュールの中で使われるのがわかるだろう。ゆえに，ライトステートなのである。

なお，現在のDirect3Dでは，3Dアクセラレーションを持ったビデオチップは，実際にはラスタ処理部分を担当するだけで，変換モジュールと照明モジュールはCPUが行っている。つまり，3Dカードを積んでいるマシンでも，K6-2の3D Now!は有効である。DirectX7からは，これらもアクセラレータで処理できるようになるらしいが，それが主流になるかどうかはわからない。また，Permediaには当初からこういった演算を行えるdeltaというエンジンが搭載されていて，OpenGLでは利用されている。

話がそれてしまったが，DrawPrimitive ()の引数を説明しよう。

```
HRESULT DrawPrimitive (
  D3DPRIMITIVETYPE dptPrimitiveType,
  DWORD  dwVertexTypeDesc,
  LPVOID lpvVertices,
  DWORD  dwVertexCount,
  DWORD  dwFlags
);
```

第1引数は，頂点並びを指定する。ポリゴンを描く場合は，以下の3つのうちからどれかを指定することになる。

D3DPT_TRIANGLELIST

先頭から3つずつの頂点で作られるポリゴンが描かれる。

D3DPT_TRIANGLESTRIP

1-2-3番めの頂点，2-3-4，4-5-6……というように，2頂点を共有したポリ

リスト1　IMでの基本ポリゴン表示

```
// Direct3Dの初期化
BOOL InitD3D( HWND hWnd )
{
    DDSURFACEDESC2 ddsd;

    // Direct3Dオブジェクトの取得
    if( lpDD->QueryInterface( IID_IDirect3D3, (void**)&lpD3D )!=DD_OK ) return InitD3DError( hWnd );

    // zバッファの作成
    memset( &ddsd, 0, sizeof(DDSURFACEDESC2) );
    ddsd.dwSize = sizeof(DDSURFACEDESC2);
    ddsd.dwFlags = DDSD_WIDTH|DDSD_HEIGHT|DDSD_CAPS|DDSD_PIXELFORMAT;
    ddsd.dwWidth = SCREENWIDTH;
    ddsd.dwHeight = SCREENHEIGHT;
    ddsd.ddsCaps.dwCaps = DDSCAPS_ZBUFFER;
    ddsd.ddsCaps.dwCaps |= (bHAL?DDSCAPS_VIDEOMEMORY:DDSCAPS_SYSTEMMEMORY);
    lpD3D->EnumZBufferFormats( *DisplayParam.GetDeviceGUID(), EnumZBufferCallback,
        (void*)&ddsd.ddpfPixelFormat );
    if( lpDD->CreateSurface( &ddsd, &lpDDZ, NULL )!=DD_OK ) return InitD3DError( hWnd );

    // zバッファをバックバッファにアタッチ
    if( lpDDSBack->AddAttachedSurface( lpDDZ )!=DD_OK ) return InitD3DError( hWnd );

    // Direct3Dデバイスの作成
    if( lpD3D->CreateDevice( *DisplayParam.GetDeviceGUID(), lpDDSBack, &lpD3DDev, NULL )!=D3D_OK )
        return InitD3DError( hWnd );

    // ビューポートの作成
    if( lpD3D->CreateViewport( &lpD3DView, NULL )!=D3D_OK )
        return InitD3DError( hWnd );

    // デバイスにビューポートを追加
    if( lpD3DDev->AddViewport( lpD3DView )!=D3D_OK )
        return InitD3DError( hWnd );

    // クリッピングボリュームの設定
    D3DVIEWPORT2 viewData;
    memset( &viewData, 0, sizeof(D3DVIEWPORT2) );
    viewData.dwSize = sizeof(D3DVIEWPORT2);
    viewData.dwX = 0;
    viewData.dwY = 0;
    viewData.dwWidth = SCREENWIDTH;
    viewData.dwHeight = SCREENHEIGHT;
    viewData.dvClipX = -1.0f;
    viewData.dvClipY = (float)SCREENHEIGHT/SCREENWIDTH;
    viewData.dvClipWidth = 2.0f;
    viewData.dvClipHeight = 2.0f*(float)SCREENHEIGHT/SCREENWIDTH;
    viewData.dvMinZ = 0.0f;
    viewData.dvMaxZ = 1.0f;
    if( lpD3DView->SetViewport2( &viewData )!=D3D_OK )
        return InitD3DError( hWnd );

    // ビューポートのセット
    if( lpD3DDev->SetCurrentViewport( lpD3DView )!=D3D_OK )
        return InitD3DError( hWnd );

    return TRUE;
}

// Direct3D初期化エラー処理
BOOL InitD3DError( HWND hWnd )
{
    ReleaseObjects();
    MessageBox( hWnd, "Initialize Error", "Direct3D Initialize", MB_OK|MB_ICONHAND );
    return FALSE;
}

// ZBuffer列挙のコールバック関数
static HRESULT WINAPI EnumZBufferCallback( DDPIXELFORMAT *pddpf, void *pddpfDesired )
{
    if( pddpf->dwFlags==DDPF_ZBUFFER ){
        memcpy( pddpfDesired, pddpf, sizeof(DDPIXELFORMAT) );
        return D3DENUMRET_CANCEL;
    }
    return D3DENUMRET_OK;
}

D3DVERTEX vertex[] = {
//    頂点              法線              テクスチャ
    {-1.0, 1.0, 0.0, 0.0, 0.0,-1.0, 0.0, 0.0 },
    { 1.0, 1.0, 0.0, 0.0, 0.0,-1.0, 0.0, 0.0 },
    {-1.0,-1.0, 0.0, 0.0, 0.0,-1.0, 0.0, 0.0 },
    { 1.0,-1.0, 0.0, 0.0, 0.0,-1.0, 0.0, 0.0 }
};

BOOL BuildScene( IDirect3D3 *lpD3D, IDirect3DDevice3 *lpD3DDev,
    IDirect3DViewport3 *lpD3DView )
{
    // 投影マトリクスの設定
    D3DMATRIX matrix;
    D3DUtil_SetProjectionMatrix( matrix, 1.0f, 1.0f, 1.0f, 50.0f );
    lpD3DDev->SetTransform( D3DTRANSFORMSTATE_PROJECTION, &matrix );
    // ビューマトリクス(いわゆるカメラ)の設定
    D3DVECTOR vUp   = { 0.0f, 1.0f, 0.0f };   // 上向きの軸
    D3DVECTOR vFrom = { 0.0f, 0.0f, -4.0f };  // 視点の位置
    D3DVECTOR vAt   = { 0.0f, 0.0f, 0.0f };   // 視線の先
    D3DUtil_SetViewMatrix( matrix, vFrom, vAt, vUp );
    lpD3DDev->SetTransform( D3DTRANSFORMSTATE_VIEW, &matrix );
    // マテリアル
    lpD3D->CreateMaterial( &lpD3DMat, NULL );
    D3DMATERIALHANDLE hmtrlMaterial;
    D3DMATERIAL       mtrl;
    D3DUtil_InitMaterial( mtrl, 1.0f, 0.5f, 0.0f );
    mtrl.dwRampSize = 16;
    lpD3DMat->SetMaterial( &mtrl );
    lpD3DMat->GetHandle( lpD3DDev, &hmtrlMaterial );
    // ライト
    D3DLIGHT          light;
    lpD3D->CreateLight( &lpD3DLight, NULL );
    D3DUtil_InitLight( light, D3DLIGHT_POINT, 0.0f, 10.0f, -5.0f );
    light.dvAttenuation0 = 1.0f;
    lpD3DLight->SetLight( &light );
    lpD3DView->AddLight( lpD3DLight );
    // レンダリングステートの設定
    lpD3DDev->SetLightState( D3DLIGHTSTATE_MATERIAL, hmtrlMaterial );
    lpD3DDev->SetLightState( D3DLIGHTSTATE_AMBIENT, 0x50505050 );
    lpD3DDev->SetRenderState( D3DRENDERSTATE_ZENABLE, 1 );
    lpD3DDev->SetRenderState( D3DRENDERSTATE_DITHERENABLE, TRUE );
    lpD3DDev->SetRenderState( D3DRENDERSTATE_SPECULARENABLE, FALSE );
    // クリップステータスの初期化
    ClipStatus[0].minx = ClipStatus[0].miny = ClipStatus[0].maxz = ClipStatus[0].minz = 0;
    ClipStatus[0].maxx = (float)SCREENWIDTH;
    ClipStatus[0].maxy = (float)SCREENHEIGHT;
    ClipStatus[1].minx = ClipStatus[1].miny = ClipStatus[1].maxz = ClipStatus[1].minz = 0;
    ClipStatus[1].maxx = (float)SCREENWIDTH;
    ClipStatus[1].maxy = (float)SCREENHEIGHT;
    // ワールド変換マトリクスの初期化
    Matrix._11 = Matrix._22 = Matrix._33 = Matrix._44 = 1.0;
    Matrix._12 = Matrix._13 = Matrix._14 = 0.0;
    Matrix._21 = Matrix._23 = Matrix._24 = 0.0;
    Matrix._31 = Matrix._32 = Matrix._34 = 0.0;
    Matrix._41 = Matrix._42 = Matrix._43 = 0.0;

    tickcount = GetTickCount();

    return TRUE;
}

void Render()
{
    // レンダリング
    lpD3DDev->BeginScene();
    // サーフェスのクリア
    D3DRECT rect = { (long)ClipStatus[side].minx, (long)ClipStatus[side].miny,
        (long)ClipStatus[side].maxx, (long)ClipStatus[side].maxy };
    if( FULLSCREEN ){
        // バックバッファのクリア
        lpD3DView->Clear2( 1, &rect, D3DCLEAR_TARGET, 0x00004488, D3DVAL(1.0), 0 );
        // zバッファのクリア
        D3DRECT rect = { (long)ClipStatus[side].minx, (long)ClipStatus[side].miny,
            (long)ClipStatus[side].maxx, (long)ClipStatus[side].maxy };
        rect.x1 = (long)ClipStatus[side^1].minx;
        rect.y1 = (long)ClipStatus[side^1].miny;
        rect.x2 = (long)ClipStatus[side^1].maxx;
        rect.y2 = (long)ClipStatus[side^1].maxy;
        lpD3DView->Clear2( 1, &rect, D3DCLEAR_ZBUFFER, 0x00000000, D3DVAL(1.0), 0 );
    } else {
        // バックバッファとzバッファのクリア
        lpD3DView->Clear2( 1, &rect, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER, 0x00004488, D3DVAL(1.0), 0
);
    }
    // クリップステータスの初期化
    D3DCLIPSTATUS status = { D3DCLIPSTATUS_EXTENTS2, 0,
        (float)SCREENWIDTH, 0.0, (float)SCREENHEIGHT, 0.0, 0.0, 0.0 };
    lpD3DDev->SetClipStatus( &status );
    // 時間管理
    DWORD count = GetTickCount();
    rot += (count-tickcount)/500.0f;
    tickcount = count;
    while( rot>=g_PI/2.0f ) rot -= g_PI;
    // フェースを回転
    D3DUtil_SetRotateXMatrix( Matrix, rot );
    // ワールド変換
    lpD3DDev->SetTransform( D3DTRANSFORMSTATE_WORLD, &Matrix );
    // レンダリング
    lpD3DDev->DrawPrimitive( D3DPT_TRIANGLESTRIP, D3DFVF_VERTEX, vertex, 4, NULL );
    // クリップステータスの取得
    lpD3DDev->GetClipStatus( &ClipStatus[side] );
    lpD3DDev->EndScene();
    // フリッピング
    if( FULLSCREEN ){
        lpDDSPrimary->Flip( NULL, DDFLIP_WAIT );
        side ^= 1;
    } else {
        lpDDSPrimary->Blt( &rcClient, lpDDSBack, &rcRect, DDBLT_WAIT, NULL );
    }
}
```

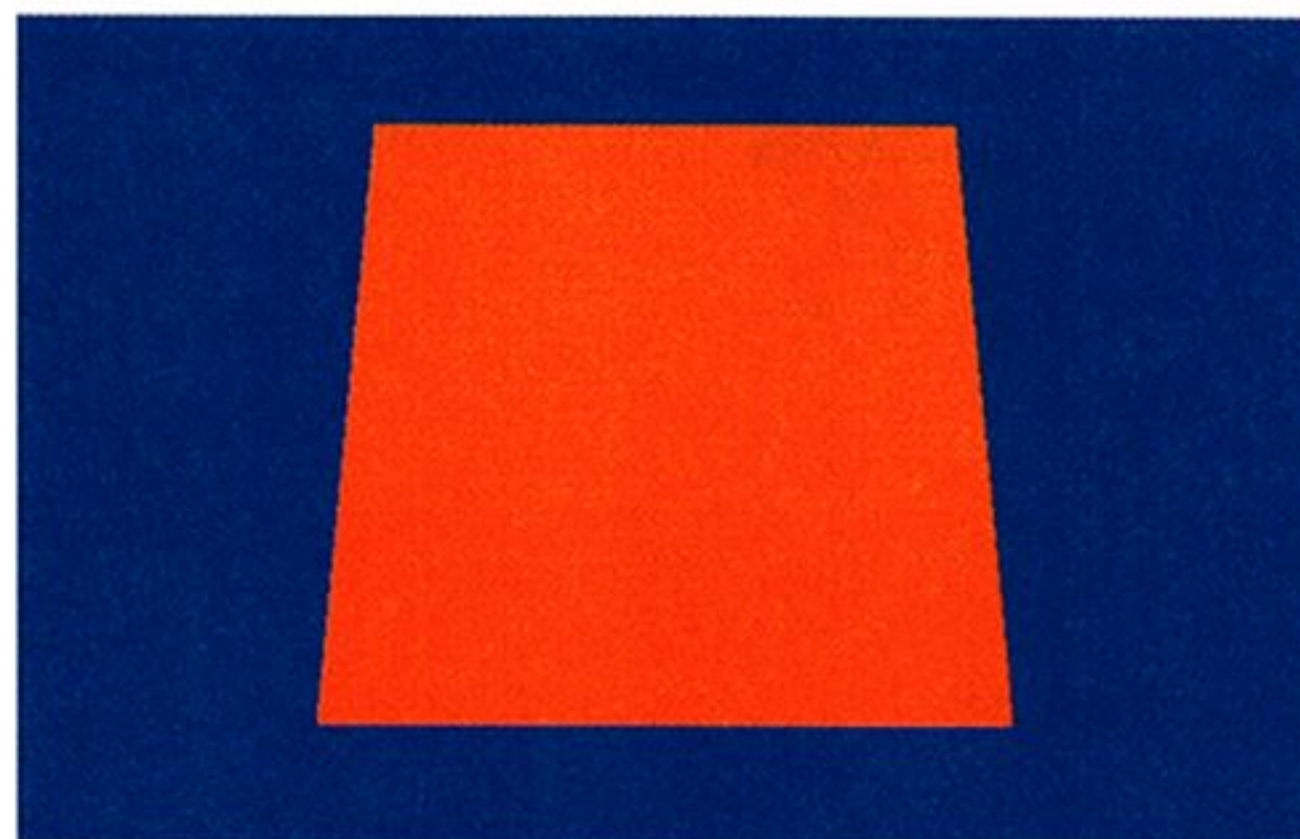



図3
とりあえずポリゴンが表示できた

ゴンが描かれる。

D3DPT_TRIANGLEFAN

2頂点を共有したポリゴンが描かれるのはD3DPT_TRIANGLESTRIPと同じだが，1-2-3，1-3-4，1-4-5……というように最初の頂点を基準として扇型に描かれる。

いい忘れたが，IMでは三角形のポリゴンしか描くことができない（点や直線は描画できる）。これを見ると，同じポリゴン数でもD3DPT_TRIANGLESTRIPやD3DPT_TRIANGLEFANを使えば頂点数を減らすことができることがわかるだろう。例によって法線は頂点に設定するので，グローシェーディングするプリミティブでエッジを立てたいときは頂点を別にするしかないが，そうでないときはできるだけD3DPT_TRIANGLESTRIPかD3DPT_TRIANGLEFANを使って倹約するべきだろう。

2番めの引数は，頂点の配列の中に入っているデータを示す。D3DFVF_VERTEXの場合，射影変換前でライティングされていない頂点座標と，法線ベクトル，テクスチャ座標のワンセットがひとつの頂点データとして設定される。次は頂点の配列だ。最初ということで，1枚の正方形のみだ。テクスチャはないのでテクスチャ座標は無視する。頂点の個数は4。最後のフラグは，クリッピングしないとか，描き終わるまで待つとかのフラグを指定できるが，普通はNULLでいい。

実行すると，ただ単に変な色の面が回っているだけだが，これがIMの第一歩だ（図3）。基本の基本だが，そんなにめちゃくちゃ難しいってわけじゃなさそうってのがわかったと思う。う〜ん，そういやこのサンプルならZバッファはいらんかったな。

テクスチャを貼る

バンプへの道，その2である。さっきのサンプルを改良してテクスチャを貼ればいいわけだが，その前にテクスチャについて。DirectX6からシングルパスマルチテクスチャがサポートされたのは話した。最大8段までのステージが用意されており，それぞれのステージにテクスチャとテクスチャオペレーションを設定することで，順番にテクスチャが処理され，さまざまな効果が期待できる。なお，ハードウェアがシングルパスマルチテクスチャに対応しているといっても，8段すべてをサポートしているとは限らない。たとえば，原稿執筆時点で唯一対応しているRivaTNTは4段までである。この辺りはD3DDEVICEDESC構造体のwMaxTextureBlendStagesを見ることで何段までサポートされているかを判断できる。また，マルチテクスチャをサポートしていないカードでは，この値が1となっている。いまからやろうとしていることは，単にテクスチャを1枚貼ろうとしているだけだ。にもかかわらず，なぜこんな話をするかというと，DirectX6からは，テクスチャが1枚だけの場合も，このテクスチャステージの1段だけを使うようにすることが推奨されているからだ。従来どおりの使い方もできるが，その場合も1段めが使われる。つまり，テクスチャステージを明示して使ったほうが，のちのち混乱することがないよ，というわけだ。

追加部分としては，テクスチャにする画像を読み込むこと，テクスチャステージを設定すること，あとは頂点配列にテクスチャ座標を入れておくことくらいだ。テクスチャの読み込みは，IMだとかなり面倒そうだが（実際かなり厄介らしい），これもお助け関数がサンプルに入っている。D3DTextrがそれだ。まず，

```
D3DTextr_CreateTexture ( "ohx.bmp" ) ;
```

などとして作成したら，

```
D3DTextr_RestoreAllTextures ( lpD3DDev ) ;
```

としてファイルからロードする。あとは，

```
D3DTextr_GetTexture ( "ohx.bmp" ) ;
```

とすれば，IDirect3DTextureインタフェイスが取得できる。らくちんだ。ちなみにこれらの関数は内部にコンテナを持っていて，複数のテクスチャをロードし，ファイル名をキーとして操作できる。必要がなくなったら，

```
D3DTextr_DestroyTexture ( "ohx.bmp" ) ;
```

として削除すればいい。さて，テクスチャステージだが，一気にやってしまおう。

```
lpD3DDev->SetTexture ( 0, D3DTextr_GetTexture
                                    ( "ohx.bmp" ) ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_ADDRESS, D3DTADDRESS_WRAP ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_MAGFILTER, D3DTFG_LINEAR ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_MINFILTER, D3DTFN_LINEAR ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_COLOROP, D3DTOP_MODULATE ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE ) ;
lpD3DDev->SetTextureStageState
        ( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE ) ;
```

それぞれの第1引数の0は，テクスチャステージ0であることを示す。つまり，いちばん最初のステージだ。SetTexture ()はそのステージに指定したテクスチャを設定するとして，問題はステートだ。

・D3DTSS_ADDRESS

ポリゴンよりもテクスチャが小さい場合の，テクスチャアドレッシングモードを設定する（図4）。いまの場合はあまり意味がない。

・D3DTSS_MAGFILTER

D3DTADDRESS_WRAP　D3DTADDRESS_MIRROR
D3DTADDRESS_CLAMP　D3DTADDRESS_BORDER
テクスチャ　プリミティブにマップ　テクスチャ　プリミティブにマップ

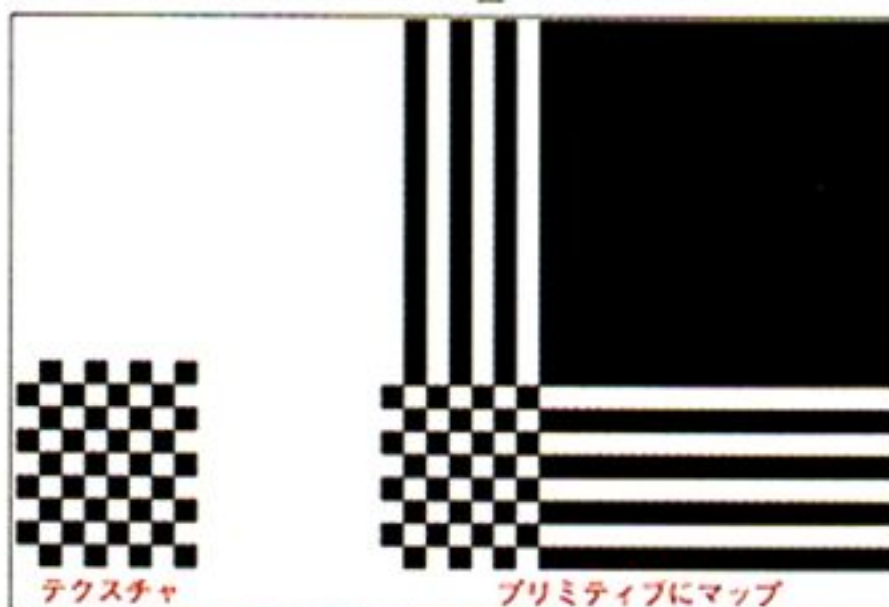
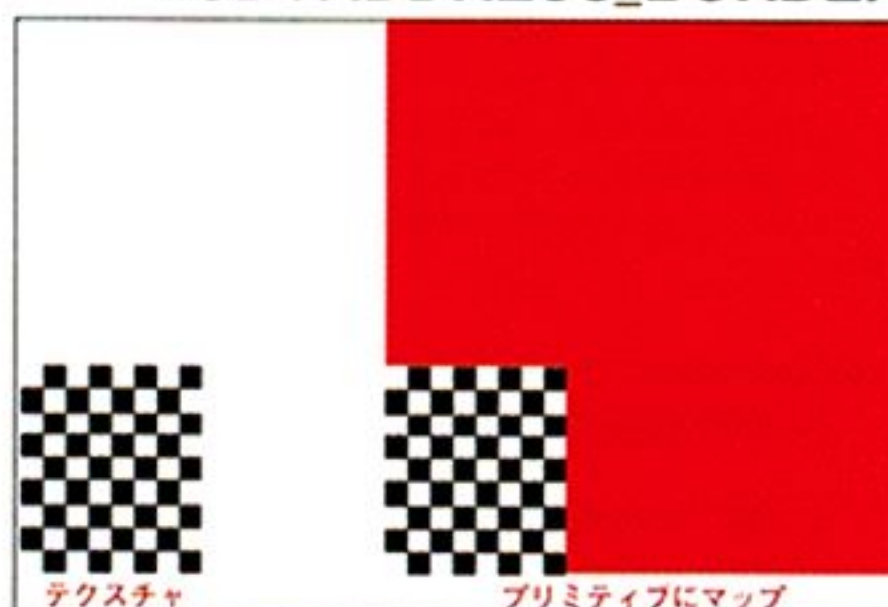



図4　テクスチャアドレッシングモード
背景色としてIDirect3DDevice3::SetTextureStageState () のD3DTSS_BORDERCOLORで設定された色が使用される

図5
IMでテクスチャを張りつけたところ

・D3DTSS_MINFILTER
テクスチャの拡大および縮小フィルタを指定する。D3DTFG_LINEARとD3DTFN_LINEARを設定するのが，いわゆるバイリニアフィルタである。
・D3DTSS_COLOROP
テクスチャカラーのオペレーションを示す。D3DTOP_MODULATEは，ARG1とARG2の乗算を行い，それをカラーとすることを示す。
・D3DTSS_COLORARG1
・D3DTSS_COLORARG2
テクスチャカラーオペレーションの引数を示す。D3DTA_TEXTUREはこのステージに設定したカラーを示し，D3DTA_DIFFUSEは照明によって陰影づけられたポリゴンのカラーを示す。
いまの例でいえば，普通に陰影づけられたポリゴンに対して，バイリニアフィルタでポリゴンのピクセルとテクセルを乗算（要するに普通にマッピング）してね，ってことだ。あとは，ポリゴンの頂点に適当にテクスチャ座標を設定してGo！割と簡単だったね（図5）。テクスチャが貼られただけで相変わらず質素だけど。なになに，ここまできたら目標はもう目の前だ。

3種のバンプマップ手法

おりゃおりゃおりゃ〜と助走をつけてみたのだが，ここでちょっと石になってしまった。ヘルプにバンプの説明が……まったくといっていいくらいない。ちょろっと列挙型メンバの説明がある程度で，こんなもんわかるか〜！　一応サンプルソースはあるが，これもなんだか……。早くも挫折のときがきたか，このまま原稿を落としてしらばっくれてるか，と思っていたら，読者（じゃないのかもしれないけど）のX-Funkさんからバンプのサンプルをいただいた（（U）氏に送られたものだけど）。やっぱ持つべきものは読者だなぁ。サンキュー，X-Funkさん。お陰でなんとなくわかったよ。
さて，Direct3Dで実現できる（あえて対応している，とはいわない）バンプマッピングには，次の3つの方法がある。

1）Emboss式

理屈は2Dグラフィックのエンボスと同じだ。ハイトマップをずらして差分を取り，エンボス処理を行ったあと，それを照明されたポリゴンの明度としてテクスチャをModulateする。ずらす方向と量は，光源から決定して各頂点のテクスチャ座標に設定する。テクスチャを駆使してバンプらしきことをやってしまおうという力技。

2）DotProduct式

各テクセルに格納された法線と，光源へのベクトルの内積を取り，それを明度としてテクスチャをModulateする。バンプマッピングとしてはいちばんまじめな方法で，とてつもなく重そうだが，実は意外に軽かったりする。Phongシェーディングもせずにそんなことができるのか？　と思うだろうが，ネタばらしはのちほど。

3）Environment式

もともとはTriTech社のPyramid3Dというチップに搭載されたバンプ方式だが，MicrosoftがDirect3Dに採用した。ライトマップと併用し，法線ベクトルからそのライトマップの参照アドレスをずらすことで，あたかも法線が振動しているかのような効果を出す。ミソは「法線ベクトルからずらすアドレスの算出方法」だが，これはテクスチャ法線のu，v要素に対し，与えられた2×2行列で変換することで得ている，ようだ。したがって，このバンプ法が適用できるのは，球などの特殊な形状に限られる。
TriTech式は特殊な場合（だけどよく使われる）にしか使えないし，SDKのサンプルではこの方式が使われているので，ここではEmboss式とDotProduct式を紹介しよう。

Emboss式バンプマッピング

まずはハイトマップを用意する。ハイトマップとは，低い部分を暗く，高い部分を明るく描いたグレースケールの画像である。照明されたポリゴン（D3DTA_DIFFUSE）からまずハイトマップを減算（D3DTOP_SUBTRACT）し，さらにその結果（D3DTA_CURRENT）からずらしたハイトマップを加算（D3DTOP_ADD），最後にテクスチャをModulateする（リスト2　Emboss式バンプマッピング）。つまりテクスチャステージは3段を必要とする。ハイトマップの加減算時には，オーバーフローやアンダーフロー（実際には頭打ち）を起こさないように注意しなければならない。ハイトマップにあまり大きな値を設定しないことと，もし対象となるポリゴンが明るめならば減算を先に，暗めならば加算を先にしよう。
さて，テクスチャステージは3段だが，テクスチャ座標はハイトマップのひとつとテクスチャは同じ座標を使うので，2つでいい。つまり，頂点データセットにテクスチャ座標をもうひとつ追加する必要がある。そうしておいて，

```
IDirect3DDevice3::SetTextureStageState
        ( dwStage, D3DTSS_TEXCOORDINDEX, dwValue ) ;
```

とすれば，dwStageで示されるテクスチャステージのテクスチャ座標として，頂点データセット内のdwValue番め（0から始まるインデックス）のテクスチャ座標が使われる。また，DrawPrimitive（）の第二引数で，今度はD3DFVF_XYZ¦D3DFVF_NORMAL¦D3DFVF_TEX2フラグを設定する。D3DFVF_XYZは頂点座標，D3DFVF_NORMALは法線ベクトル，D3DFVF_TEX2はテクスチャ座標が2セットあることを示している。ちなみに，さっきのテクスチャサンプルで登場したD3DFVF_VERTEXは，D3DFVF_XYZ¦D3DFVF_NORMAL¦D3DFVF_TEX1のマクロである。
ちょっと面倒なのは頂点のテクスチャ座標の算出だ。これは光源とプリミティブの位置関係が変わったときに毎回行う必要がある。平行光源にすれば，すべての頂点で同じずれを設定すればいいのでちょっと楽なのだが，ここではあえて点光源にしてみた。まず光源の座標をプリミティブのローカル座標に変換する。これには，ワールド変換の逆変換（つまりワールド変換マトリクスの逆行列で変換）を行えばよい。
逆変換マトリクスの作成および頂点の変換は，D3DMath内のお助け関数D3DMath_MatrixInvert（）およびD3DMath_VectorMatrixMultiply（）で行える。こうすることで，光源から頂点までのベクトル，つまり光の向きをローカル座標で取得できる（図6）。あとはそれを単位長にしておいて，xおよびz要素をテクスチャのuおよびvのずれとしてやる。これを各頂点に対して行えば完成だ。
テクスチャが少々ごちゃごちゃしていて，バンプがわかりにくいので，テクスチャを貼らない実行例も示しておく（図7）。ちょっとずれを大きくしす

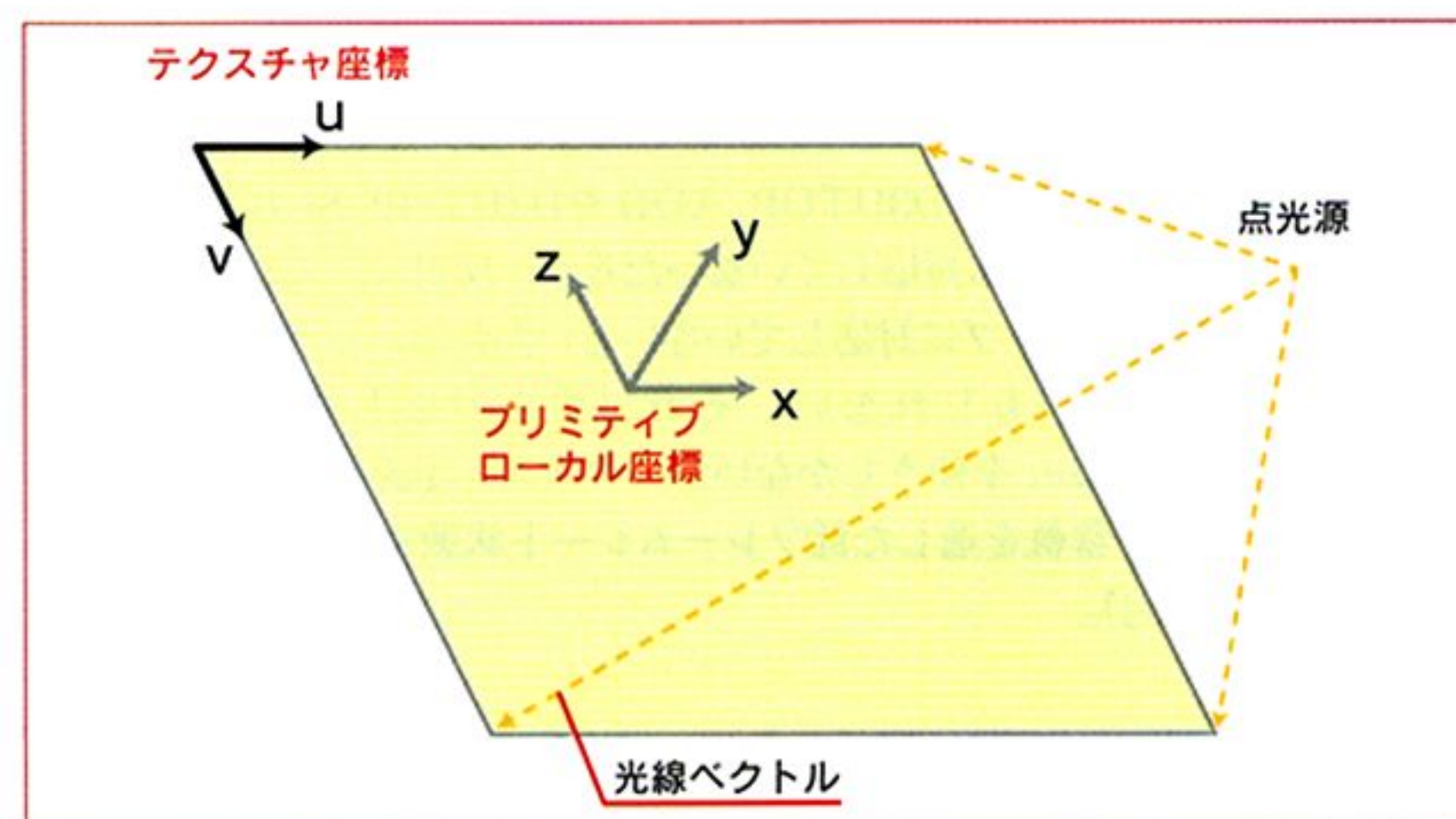



図6　テクスチャ座標のずれの算出

リスト2　Emboss式バンプマッピング

```
BOOL BuildScene( IDirect3D3 *lpD3D, IDirect3DDevice3 *lpD3DDev,
    IDirect3DViewport2 *lpD3DView )
{
            :
     (中略)
            :
    // テクスチャ
    lpD3DDev->SetTexture( 2, D3DTextr_GetTexture( "earth.bmp" ) );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_TEXCOORDINDEX, 0 );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_MAGFILTER, D3DTFG_LINEAR );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_MINFILTER, D3DTFN_LINEAR );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_COLOROP, D3DTOP_MODULATE );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_COLORARG1, D3DTA_CURRENT );
    lpD3DDev->SetTextureStageState( 2, D3DTSS_COLORARG2, D3DTA_TEXTURE );
    // バンプマップ
    // ずらしたあとのバンプテクスチャ
    lpD3DDev->SetTexture( 1, D3DTextr_GetTexture( "earthbump.bmp" ) );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 1 );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_MAGFILTER, D3DTFG_LINEAR );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_MINFILTER, D3DTFN_LINEAR );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_CURRENT );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_TEXTURE );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_ADDRESS, D3DTADDRESS_CLAMP );
    // ずらしたあとのバンプテクスチャからずらす前のものを引く
    lpD3DDev->SetTexture( 0, D3DTextr_GetTexture( "earthbump.bmp" ) );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_MAGFILTER, D3DTFG_LINEAR );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_MINFILTER, D3DTFN_LINEAR );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLOROP, D3DTOP_SUBTRACT );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_DIFFUSE );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_TEXTURE );
    // 以降のステージの無効化
    lpD3DDev->SetTextureStageState( 3, D3DTSS_COLOROP, D3DTOP_DISABLE );
            :
     (中略)
            :
}

void ApplyBump( BUMPVERTEX *vertex, int num, D3DMATRIX& matrix, LPDIRECT3DLIGHT lpLight )
{
    D3DLIGHT light;
    D3DVECTOR position, direction;
    light.dwSize = sizeof(D3DLIGHT);
    lpLight->GetLight( &light );
    // 逆行列
    D3DMATRIX invmatrix;
    D3DMath_MatrixInvert( invmatrix, matrix );
    // ライトをワールド座標からプリミティブのローカル座標へ
    D3DMath_VectorMatrixMultiply( position, light.dvPosition, invmatrix );
    for( int i=0; i<num; i++ ){
        // 各頂点に対する光線の向き（点光源の場合）
        direction.dvX = vertex[i].v.x-position.x;
        direction.dvY = vertex[i].v.y-position.y;
        direction.dvZ = vertex[i].v.z-position.z;
        // 単位ベクトル化
        Normalize( direction );
        // バンプのコーディネイトをずらす
        vertex[i].tu2 = vertex[i].v.tu+direction.x*0.03f;
        vertex[i].tv2 = vertex[i].v.tv-direction.z*0.03f;
    }
}

void Normalize( D3DVECTOR &v )
{
    D3DVALUE len = (D3DVALUE)sqrt(v.x*v.x+v.y*v.y+v.z*v.z);
    v.x /= len;
    v.y /= len;
    v.z /= len;
}
```

ぎたかもしれない。この方法ならば，仮にハードウェアがシングルパスマルチテクスチャに対応していなくても，理論的にはマルチパスマルチテクスチャ（ちょっとくどいな）で処理できるかもしれないが，どの道マルチテクスチャに対応していなければD3DTOP_ADDやD3DTOP_SUBTRACTといったオペレーションにも対応していないだろう。反対に，これらに対応していれば，一般的にバンプに対応していないといわれるハードウェアでもレンダリングできるかもしれない。それにすら対応していなければ，Reference Rasterizerを使うしかないが，ロスユニ4話なので覚悟すること（ロスユニ4話：常軌を逸した低フレームレート状態を示す比喩。動詞形は「ヤシガニ居る」）。

DotProduct式バンプマッピング

この方式は，D3DTOP_DOTPRODUCT3というテクスチャオペレーションを使う。このオペレーションは次の式で表される。

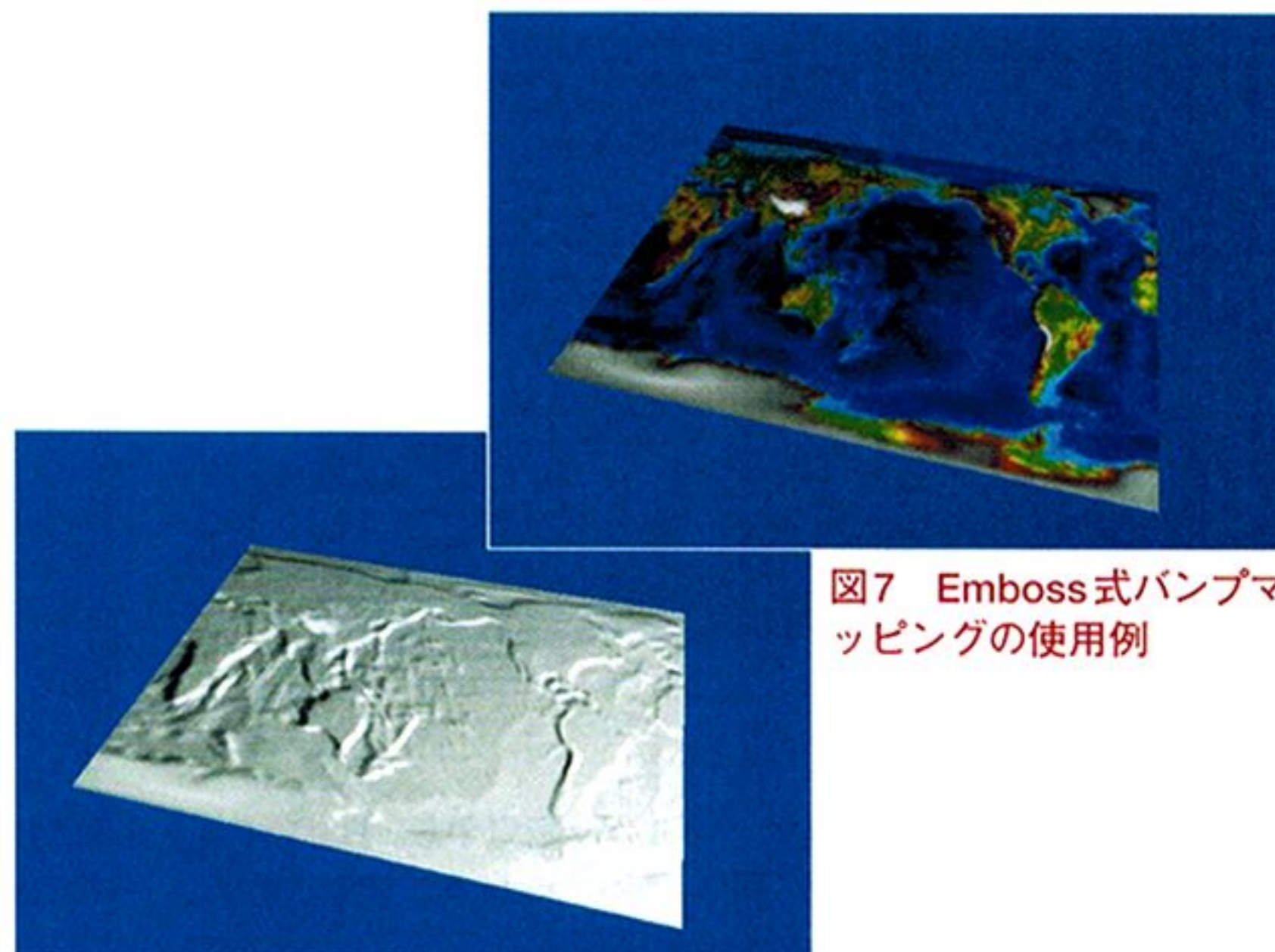



図7　Emboss式バンプマッピングの使用例

$$Srgba = Arg1r \times Arg2r + Arg1g \times Arg2g + Arg1b \times Arg2b$$

ここで，Srgbaは出力されるRGBA色，Arg1とArg2はオペレーションの引数であり，サフィックスはそれぞれR，G，Bの要素であることを示す。この式を見て，なにか気づかないだろうか。それぞれの引数のR，G，Bにベクトルのx，y，zを入れれば，Srgbaにはそのベクトルの内積が出力されるのだ。つまり片方の引数にはRGB値に法線のxyzを入れたテクスチャを指定し，もうひとつは光源の法線を指定すれば，明度を得ることができるのだ。

まずは法線ベクトルの入ったテクスチャを作る必要がある（リスト3）。ハイトマップならばグラフィックツールで作ることは難しくないが，ベクトルをRGBに分けて格納しろなんてのは，グラフィックツールでは無理だ。したがって，まずはハイトマップを読み込み，そこから法線ベクトルのテクスチャを生成する。xおよびz方向のベクトルは，テクスチャu方向およびv方向のハイトの差分から計算しているが，この計算をもう少し考えれば最終的な品質は上がるかもしれない。

ここで，ベクトルの要素が−1.0から1.0までの範囲に対して，RGB値が0〜255になるように設定する。上の式とつじつまがあわなくなってくるが，上の式は理論的な話であって，実際にこれで正しく明度が計算されるようにできているのだ。

テクスチャの法線はできた。今度は光源の法線である。平行光源であれば，引数としてD3DTA_TFACTORを指定する手もある。これは，IDirect3DDevice3::SetRenderState（）でD3DRENDERSTATE_TEXTUREFACTORとして指定した色を使うというものだ。ここで色の代わりに光線のベクトルを指定すればよい。ただ，今回も点光源にしてみようと思う。その場合はどうするか。ここで，シェーディングを考えてみよう。まずライトの位置から頂点の色が算出され，レンダリング時にそれが補間されてピクセルカラー（DIFFUSE）となる。つまり，頂点の色の代わりにその頂点における光線ベクトルを設定できれば，DIFFUSEにはそれを補間した各ピクセルにおける光線ベクトルが入ることになる。

しかし，そんなことができるのだろうか。できるのである。Direct3Dは，プログラム側でライト計算を自前で行った場合を考えて，「照明を行ったあとの頂点」というものを設定できるのだ。この場合，頂点データ内の法線ベクトルは必要なくなるので，頂点データセットの法線ベクトルの代わりに頂点の照明後のカラーを加え，DrawPrimitive（）のフラグにはD3DFVF_NORMALの代わりにD3DFVF_DIFFUSEをセットする。座標さえ決めれば光源を作る必要もなくなるのだが，まあそれはそのままでいいだろう。

頂点の光線ベクトル（正確にはその逆向き）の算出は，サンプルEmbossと似ているが，今度は法線テクスチャと同様にレンジを0〜255に変換し，頂点カラーとして設定している。

完成したのがサンプルDotProductだ（図8）。Emboss同様，テクスチャなしの実行例も示す。Embossよりもバンプの質は高い。また，もしグローシェーディングを行う場合でも，最初の法線テクスチャを作る段階で頂点

ベクトルを考慮すれば，それなりの精度でレンダリングできるはずだ。

バンプ常用不可

しかしまあ，なんだな。こんなんででかい顔して「バンプ対応」なんていってほしくないもんだな。いや，でかい顔はしてないのか，ヘルプにろくな説明ないし。結局プログラム側でやることが多くて，バンプに対応したカードがあっても，CPUの処理時間がばかにならない。今回のサンプルでは頂点は4つしかなかったから（頂点の法線ベクトル算出部分は）どうってことないが，頂点が増えるとするするというわけにもいかないだろう。手を抜きゃいいんだろうが。やっぱハイトマップをぱくっと食べさせれば，あとは全自動って感じじゃなくちゃ，本当に対応したとはいえないんじゃなかろうか。これじゃ一発芸にしかならないよ。

まあでも，IM結構楽しいよ。今回は結局フレーム構造みたいなことはやらなかったけど，お助け関数にそれらしきものがあるみたいだし。あと，Xファイルを読むサンプルもあるから，次回はその辺を予告しておこうか。

図8　DotProduct式バンプマッピングの使用例

リスト3　Dot Product式バンプマッピング

```
LPDIRECT3DTEXTURE2 InitBumpMap( LPDIRECTDRAWSURFACE4 pddsBumpSrc )
{
    DDSURFACEDESC2 ddsd;

    // バンプサーフェス
    D3DUtil_InitSurfaceDesc( ddsd );
    pddsBumpSrc->GetSurfaceDesc( &ddsd );

    ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH | DDSD_PIXELFORMAT;
    ddsd.ddsCaps.dwCaps = DDSCAPS_TEXTURE | (bHAL?DDSCAPS_VIDEOMEMORY:DDSCAPS_SYSTEMMEMORY);
    ddsd.ddsCaps.dwCaps2 = 0L;
    ddsd.ddpfPixelFormat.dwSize = sizeof(DDPIXELFORMAT);
    ddsd.ddpfPixelFormat.dwFlags = DDPF_RGB;
    ddsd.ddpfPixelFormat.dwRGBBitCount        = 24;
    ddsd.ddpfPixelFormat.dwRBitMask      = 0xff0000;
    ddsd.ddpfPixelFormat.dwGBitMask      = 0x00ff00;
    ddsd.ddpfPixelFormat.dwBBitMask      = 0x0000ff;

    LPDIRECTDRAWSURFACE4 pddsBumpMap;
    LPDIRECT3DTEXTURE2   ptexBumpMap;

    if( lpDD->CreateSurface( &ddsd, &pddsBumpMap, NULL )!=DD_OK )
        return NULL;

    while( pddsBumpSrc->Lock( NULL, &ddsd, 0, 0 ) == DDERR_WASSTILLDRAWING );
    DWORD lSrcPitch = ddsd.lPitch;
    BYTE* pSrc      = (BYTE*)ddsd.lpSurface;

    while( pddsBumpMap->Lock( NULL, &ddsd, 0, 0 ) == DDERR_WASSTILLDRAWING );
    DWORD lDstPitch = ddsd.lPitch;
    BYTE* pDst      = (BYTE*)ddsd.lpSurface;

    for( DWORD y=0; y<ddsd.dwHeight; y++ ){
        BYTE* pDstT = pDst;
        BYTE* pSrcB0 = (BYTE*)pSrc;
        BYTE* pSrcB1 = ( pSrcB0 + lSrcPitch );

        if( y == ddsd.dwHeight-1 )   // Don't go past the last line
            pSrcB1 = pSrcB0;

        for( DWORD x=0; x<ddsd.dwWidth; x++ ){
            LONG  v00 = *(pSrcB0+0); // カレントピクセルのハイト
            LONG  v01 = *(pSrcB0+4); // 右のピクセルのハイト
            LONG  v10 = *(pSrcB1+0); // 下のピクセルのハイト

            LONG iDu = v00-v01;      // u方向の差分
            LONG iDv = v00-v10;      // v方向の差分

            // RGBに法線ベクトルを格納
            // (r,g,b) = (x,y,z) = (iDu,1,-iDv)
            // -128 <= x,y,z <= 127
            // 負数を表すためのバイアスは128
            int x = iDu;
            int y = 127;
            int z = -iDv;
            // 単位ベクトル化
            float len = (float)sqrt(x*x+y*y+z*z);
            x = (int)(x/len*127.9f);
            y = (int)(y/len*127.9f);
            z = (int)(z/len*127.9f);
            *pDstT++ = (BYTE)(z+128);
            *pDstT++ = (BYTE)(y+128);
            *pDstT++ = (BYTE)(x+128);

            pSrcB0 += 4; // ピクセルを進める
            pSrcB1 += 4;
        }
        pSrc += lSrcPitch; // 次のラインへ
        pDst += lDstPitch;
    }
    pddsBumpMap->Unlock(0);
    pddsBumpSrc->Unlock(0);

    // テクスチャの取得
    if( pddsBumpMap->QueryInterface( IID_IDirect3DTexture2, (void**)&ptexBumpMap )!=D3D_OK )
        ptexBumpMap = NULL;

    pddsBumpMap->Release();

    return ptexBumpMap;
}

BOOL BuildScene( IDirect3D3 *lpD3D, IDirect3DDevice3 *lpD3DDev,
    IDirect3DViewport2 *lpD3DView )
{
            :
     (中略)
            :
    // テクスチャ
    lpD3DDev->SetTexture( 1, D3DTextr_GetTexture( "earth.bmp" ) );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_TEXCOORDINDEX, 0 );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_MAGFILTER, D3DTFG_LINEAR );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_MINFILTER, D3DTFN_LINEAR );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_MODULATE );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_CURRENT );
    lpD3DDev->SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_TEXTURE );
    // バンプマップ
    lpD3DDev->SetTexture( 0, lpD3DBump );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_TEXCOORDINDEX, 0 );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_MAGFILTER, D3DTFG_LINEAR );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_MINFILTER, D3DTFN_LINEAR );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLOROP, D3DTOP_DOTPRODUCT3 );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_TEXTURE );
    lpD3DDev->SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_DIFFUSE );
    // 以降のステージの無効化
    lpD3DDev->SetTextureStageState( 2, D3DTSS_COLOROP, D3DTOP_DISABLE );
            :
     (中略)
            :
}

void GenerateLightVector( BUMPVERTEX *vertex, int num, D3DMATRIX& matrix, LPDIRECT3DLIGHT lpLight )
{
    D3DLIGHT light;
    D3DVECTOR position, direction;
    light.dwSize = sizeof(D3DLIGHT);
    lpLight->GetLight( &light );
    // 逆行列
    D3DMATRIX invmatrix;
    D3DMath_MatrixInvert( invmatrix, matrix );
    // ライトをワールド座標からプリミティブのローカル座標へ
    D3DMath_VectorMatrixMultiply( position, light.dvPosition, invmatrix );
    for( int i=0; i<num; i++ ){
        // 各頂点に対する光源の向き（点光源の場合）
        direction.dvX = position.x-vertex[i].v.x;
        direction.dvY = position.y-vertex[i].v.y;
        direction.dvZ = position.z-vertex[i].v.z;
        Normalize( direction );
        int x = 128+(int)(direction.x*127.9);
        int y = 128+(int)(direction.y*127.9);
        int z = 128+(int)(direction.z*127.9);
        vertex[i].color = (((x<<8)|y)<<8)|z;
    }
}
```

言語処理プログラムを作る　第1回

石上達也　Ishigami Tatsuya

現在では，最初から開発環境が揃っている環境も多かったり，市販のツールで間にあうためか，プログラム言語の制作というのは，あまり行われていません。半面，ツールでのマクロ処理など，内部インタプリタなどの必要な場面はむしろ増えているのかもしれません。今回からX-BASICライクでポータブルなインタプリタ制作を行っていきます。

プログラムを作る際，まず必要になるもの。それは，開発ツールです。もちろん，PC用の開発ツールは，パソコンショップへ行けば，いろいろ売っています。今回，紹介するプログラムも，新宿のパソコンショップで買ってきた，マイクロソフトのVisual C++で作成しています。

市販ソフトとかが揃っている場合はそれでいいでしょう。では，新しいCPUのプログラムは，どうやって作ればよいでしょうか？

最低限の開発ツールは，CPUメーカーがとりあえず出す，というのが慣例になっているようですが，これは，必ずしも使い勝手がよいものではなかったり，極端に高価なものだったり，大口顧客にしか配布されないものだったり，変な守秘義務項目のせいで成果が公に発表できなかったりと，一般ユーザー(というか非商業ベースの我々アマチュア)にはなかなか手が出しづらいようです。

CPUメーカーとは別に，開発ツールの開発・販売・サポートを行っているソフトハウスもありますが，インテルのx86系，モトローラの68K系以外の開発ツールとなると市場が極端に狭まってしまうので，その分，ツールの値段に跳ね返り，個人では払えないような金額のものになってしまいます。それ以外のCPUというのは，組み込み用途が主体なわけで，主な顧客層は法人ですから，まぁ，数十万円という金額は，そんなに高価なものではないかもしれません。

この状況は別にいまに始まったわけではなく，10年前からずっと続いています。

当時，多くの人は，アマチュアの作ったアセンブラを使用していました。たいてい，これらは最初の開発ツールでしたから，利用できるツールなどそれ以前には存在せず，後述するアセンブラの機能をまったく使わず，人手でこつこつ作られていたようです。

それらのアセンブラは雑誌に発表され，読者によって，改良・移植され，ときとして，メーカー製の数十万円するものよりも高性能・高機能ということもありました。

表1　2進数と16進数の対応

2進数	16進数	10進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	a	10
1011	b	11
1100	c	12
1101	d	13
1110	e	14
1111	f	15

最近は，マイクロソフトから安価で強力な開発ツールが簡単に手に入るようになったためか，開発システムを作る人も減ってきましたが，世の中には，まだまだマイクロソフトが助けてくれない，わくわくするような分野もあるわけで，そういう分野では，ないモノは自分で作ろう，という10年前のDIYの精神が必要となってきます。

個人的な意見ですが，現在のC++やBASICは必ずしもMMXや3D Now!命令をサポートするのに最適な言語ではないと思っています。いままで，CPUは高級言語をサポートしやすいように進化してきましたが，これらの命令は，DSP的な発想で実装されていて，高級言語からどう扱われるか，説得力のある結論はまだ出ていません。今度は，高級言語のほうから歩み寄る必要があると思っています。

CやJavaのように成熟した分野で，大手メーカーの市販品に対抗するのは，かなりしんどいのですが，コンピュータの世界は，新しい動きがまだまだあり，そんな世界に飛び込む人に，この記事がお役に立てば，と思っています。

＊1 Digital Signal Processor：CPUに比べ，数値演算を低消費電力・安価で行えるプロセッサ

この連載で扱うもの

本誌で，

① マニュアルさえあれば，どんなCPUでも，最低限のプログラミング環境を整えられる

② メーカーサポートが突然打ち切られても，なんとかやっていけるだけの情報を共有することを目標にしましょう。

具体的には，

・インタプリタ型の電卓プログラム
・中間コード型のコンパイラ
・中間コード型のBASIC

最終的には，DOS Boxで走る(画面・音源制御など特定のハードウェアに依存した部分を持たない)X-BASICライクな言語を作れたらいいなぁと思っています。

このなかには，

・文字列の切り出し(形態素分析)
・式の評価
・ラベルの管理(文字列と数値の対応付け)
・コンパイラの作成
・中間コードの実行

などが含まれ，プログラミング言語処理に必要な技術がひととおり含まれています。「中間コードの実行」とは，ある仮想のコードを1命令ずつ解釈し，ソフトウェアで実行をシミュレートすることです。この仮想的なコードの代わりに，68060とかPentiumとか実存するCPUのコードを用いれば，そのまま，デバッガやシミュレータとなります。

また，「最適化」と呼ばれる技術をプログラミング言語処理の範疇に入れる人もいますが，特定の構成に強く依存したどろどろした技術です。無理に一般化して説明すると，抽象的なお話になってしまいますので，この連載では扱いません(逆に，最適化の必要ない中間コードを考えます)。

プログラミング言語とは

当たり前のことですが，デジタルコンピュータは，0と1の入力に基づいて，動作します。どんなに単純な動作，たとえば，1と2の和を求めるという場合でも，

```
10111000  00000001
00000000  00000000
00000000  10000011
11000000  00000010
```

というプログラムが必要です(Pentiumの場合。あっているかな？)。通常は，0と1の2進数4桁を1文字で表せるようにして，表1のような16進数(2の4乗で16ね)を用いて，

```
0xb801
0x0000
0x0083
0xc002
```

と扱います。これでも扱いづらいので，上の数値列を扱う場合，

```
mov     eax,1
add     eax,2
```

というように少しはわかりやすい形で扱います(1行目，eaxレジスタに1を代入。2行目eaxレジスタに2を足す)。この表記がアセンブリ言語(Assembly Language)と呼ばれ，010101……列をマシン語(Machine Language，あるいはMachine Codeと呼ばれるほうが多い)と呼ばれます。基本的に，アセンブリ言語とマシン語は，1対1で対応して，単純な置き換え作業で，変換が可能です。単純な置き換え作業は，コンピュータのもっとも得意とする処理ですから，人の手でなく，コンピュータで行います。このアセンブリ言語→マシン語の変換を行うプログラムをアセンブラ(Assembler)，マシン語→アセンブリ言語の変換を行うプログラムを逆アセンブラ(Dis-assembler)と呼びます。

この表記方法は，プログラムを1行ごとに追えば，CPUの動作を完全に把握できます。ですから，1＋2を行うのに，

・メモリが何バイト必要で，
・計算時間は何ナノ秒で，
・メモリアクセスはどの順序で行われるのか，

ということが，ちゃんとCPUの技術資料に書いてあるはずです。これは，ある分野では重要なのですが，よくよく考えると，1と2の和がいくつになるかを求めるの場合は，そんなに重要なことではありません。逆に，CPUの挙動をいちいち考慮しながら，プログラミングをするよりは，

　1＋2

とキー入力すれば，ただちに

　3

と表示するほうが，たとえ，計算結果に何ミリ秒かかるのかわからなくても便利です。

このようにプログラミング言語といっても，アセンブリ言語の様にCPUの挙動に近いものと，人間の表記に近いものがあるのですが，前者を低級言語といい，後者を高級言語と呼びます。この低級・高級というのは，相対的な見方ですので，例えば，アセンブリ言語から見れば，C言語は高級言語ですが，JavaやTclから見れば低級言語と呼べなくもありません。

プログラミング言語に必要なもの

アセンブリ言語とマシン語の変換の他にも，以下のような必要な機能があります。

●アドレス(ラベル)の扱い

たとえば，「画面に"こんにちわ"という文字を表示する」という動作は，CPUで直接理解できませんから，以下のように噛み砕いて，与えてやります。

a) 0x3248a0というメモリの場所(Address)に"こんにちは"という文字列を格納する。
b) 0x432b48という場所に文字列を表示するプログラム(サブ・ルーチン)があるので，a)の文字列を指定して実行する。

悪いことに，これらの場所情報は，プログラム開発中に，ころころ変わってきますので，直接数値で指定するととても不便です。たとえば，a)はstringHello，b)は，PrintStringというように名前(ラベル)をつけて，値が変わるたびに自動的に更新されたほうが便利です。

●表記の変換(式の評価)

前述のように，CPUの動作を規定するアセンブリ言語は，我々の日常用いる表記方法とは異なります。特に，我々が親しんでいる，

　1＋2×3×(4＋5)

のような算用表記は，最低限扱える必要があります。[*2]

＊2　一時，この方法とは異なる表記を用いたForthというプログラミング言語がありましたが，いまではほとんど見かけないようです。

●制御命令

コンピュータに計算を指示する場合，

「もし，A(という条件)が成り立つなら，B(という動作)を実行しろ」

とか，

「A(という条件)が成り立つ間は，B(という動作)を実行し続けろ」

という方法が使えると便利です。C言語やBASICでいうと，前者がif文，後者がwhile文ですね(残念ながら，日本語ではなく，英語ですが)。

しかし，アセンブリ言語にあるのは，CPU内のフラグと呼ばれる情報に基づいて，

① フラグが0だったら，指定されたアドレスを実行
② フラグが1だったら，指定されたアドレスを実行

のように，プログラムの流れを変えるようなものばかりです。CPUによっては，フラグ(プロセッサステータスレジスタだっけ？)の状態により，1命令を実行するか，スキップするか選べるものもありますが，アセンブリ言語での1命令では，あまりたいしたことはできません。

そこで，先ほどのif文やwhile文を①や②のレベルまで，噛み砕いてCPUに理解できるよう変換します。

今月のテーマ

というわけで，今月は，準備体操として，形態素分析と式の評価を行うための簡単な電卓プログラムを紹介します。プログラムの作り方を説明するのが目的ですから，電卓プログラムといっても，Windowsのアクセサリにあるような，見栄えのよいものではなく，DOS Boxで動く地味なものです(写真1)。ただし，このプログラムには，「形態素分析」と「式の評価」という言語処理系を作成するうえで，重要な要素が2つ入っています。

もう10年も前のことになりますが，大先輩ライターの瀧山氏に「式の評価」方法を教えていただいただけで，中学生だった私は，自力でBASICコンパイラを作れるようになりました(発表したのは高校生になってから)。人それぞれ，得意，不得意分野は違うでしょうが，個人的には，言語処理系を作る場合，この「式の評価」がいちばん重要で，唯一自力では解決できないところでした。

「式」とはなにか？

式は，数値と演算記号から成り立っています。さらに細かくいうと，数値とは数字が複数並んだものです。たとえば，1999という数値は，「1」という数字と3つの「9」という数字から構成されています。言葉遊びのようですが，ここで，数値と数字とは，場合によっては，違う意味を持つことに注意してください。

今回作成したプログラムで，演算子とは，四則演算(＋，－，×，／)としますが，まず，説明のために，加算・減算のみを扱い，あとで，加算，除算を追加します。

ここで，もう一度「式」とはなにかということを整理すると，

　数値 ＋ "＋"あるいは"－"＋ 数値

ということになります。

世の中には，こういう「式」とか「言語」を定義する際に，以下のようなよく使われる記号があります。

| あるいは
[ ] あってもなくてもよい（オプション）
{ } 繰り返し（1～無限回まで）

これを使うと，

＜式＞ = ＜数値＞ ＜加減算＞ ＜数値＞
＜加減算＞ = "+" | "-"
＜数値＞ = {0|1|2|3|4|5|6|7|8|9}

と表せます。実際には，1+2+3や1+2+3+4という式も存在するので，

【定義1】

＜式＞ =＜数値＞ [{ ＜加減算＞ ＜数値＞ }]
＜加減算＞ =＜"+" | "-"＞
＜数値＞ = { 0|1|2|3|4|5|6|7|8|9}

と定義できます。これをプログラムにしたものが，リスト1です。このプログラムは，実行するとキーボードからの入力待ちになりますので，たとえば，

Expression > 1+2+3+4+5+6+7+8+9+10

と入力すると，

Answer > 55

と出力します。

ポインタとバックトラック

厳密にいうと，コンピュータは，"1＋2×3"のような式を直接扱えません。一度，これらを文字列として認識し，1文字ごとに順を追って処理します。そこで，プログラムは，「現在，文字列の中のどの文字に注目しているか」を表す情報が必要になってきます。通常，この情報は，ポインタ(Pointer)変数と呼ばれています。

一般に，文字列は左側の文字からコンピュータのメモリへ格納されることになっているので，ポインタもいちばん左側の文字から，順に右側に動かしたほうが，スマートにプログラム処理できます。

ところが，プログラムで処理しやすい文法が，人間の扱いやすい文法かというと，そんなことはなく，両者がともに成立しない場合は，後者を優先させたほうが優れたプログラムといえるでしょう。

たとえば，C言語で，

void main(int argc, char ＊argv[ ])

と左から読んでいっても，これではなにを意味するのかはっきりしません。次の文字を見て，

a）"；"なら関数のプロトタイプ宣言
b）"{"なら関数本体の定義

リスト1　加減算をサポートした電卓

```
//
// 加減算の処理を行うプログラム
//

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

void  main(void);
double    ProcessExpression(void);
double    ProcessNumerics(void);
int       Amatch(char*);

char  *tokenPtr;    /*トークン分析のためのポインタ*/
char  inputBuff[300];

void
main(void)
{
   while(1){
      printf("Expression> ");
      gets(inputBuff);
      tokenPtr = inputBuff;
      // 何も入力されなければ、終了
      if(tokenPtr[0] == '\0') break;

      printf("Answer> %f\n", ProcessExpression());
      if(tokenPtr[0]){
         // <式>の後ろに、文字がある
         printf("Syntax Error\n");
      }
   }
}

/*
** <式>の評価
*/
double
ProcessExpression(void)
{
   double value = ProcessNumerics();         /* <数値>の収得 */

   while(1) {
      if(Amatch("+")){
         value = value + ProcessNumerics();
      } else if(Amatch("-")){
         value = value - ProcessNumerics();
      } else {
         break;
      }
   }
   return(value);
}


/*
** <数値>の評価
*/
double
ProcessNumerics(void)
{
   double value;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   if(isdigit(*tokenPtr)) {
      value = atof(tokenPtr);
      /* 読み終わった数字と小数点を飛ばす */
      while(isdigit(*tokenPtr) || *tokenPtr == '.') tokenPtr++;
   } else {
      /* 数字でない */
      value = 0;
      printf("Syntax Error\n");
   }
   return(value);
}

/*
** 与えられた文字列を切り取れれば、文字列文tokenPtrを進め、 1を返す
** 切り取れなければ、tokenPtrを進めずに、0を返す
*/
int
Amatch(char *s)
{
   char   *saveToken = tokenPtr;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   while(*s++ == *tokenPtr++) {
      if(*s == '\0') {
         /* 比較する文字列の最後まで達した */
         return(1);
      }
   }
   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(0);
}
```

COLUMN ポインタの扱い

私の経験では，このポインタの移動を統一して扱えるかどうかというのが，言語処理プログラムを作成するひとつの鍵になると思っています。世の中すべてがうまくいくわけではありませんが，私は，

- 次に注目する文字をポインタは指す
- ポインタの進め方は，ポストインクリメント

の2点をできるだけ守れるように心がけています。

今月のプログラムでは，ポインタを，

```
char ＊tokenPtr;
```

という変数で扱っています。バックトラックが発生しない文法であれば，文字列を一度認識したあとで参照する必要はありませんから，認識された文字列の分だけ，すぐにポインタを進めることが可能です。これらの2つの動作を組み合わせて，

1）tokenPtr からの文字列を，与えられた文字列を比較
2-a) 文字列があっていれば，その分 tokenPtr を進め，TRUE を返す。
2-b) あっていなければ，何もせず FALSE を返す

という関数，

```
BOOL Amatch(char ＊s);
```

を使っています(Advance if matched の略)。

この関数は，以前 Small-C という C コンパイラを移植中に見つけたものですが，名前が気に入っていて，それ以来，SX-BASIC とか「来来仏語」などにも使わせていただいています。

実際の処理系では，バックトラックを発生する処理もあるので，

```
BOOL Match(char ＊s);
```

という，tokenPtr を進めない関数や，文字列が合致後，さらに長い文字列に合致していないかを含めて考慮する関数，

```
BOOL  AmatchS(char ＊s);
(Amatch with Separator)
(s="func1"の場合，func1( )にTRUE を返しても，func12( )にはFALSE を返す)
```

などのバリエーションを作っています。

プログラミング言語を作成する際には，yacc とか Meta などのコンパイラ・コンパイラと呼ばれるツールを使用する場合もありますが，上記の2点を守っていれば，これらのツールが特に必要だとは思いません。いまだったら，逆に Visual Studio に統合されていない分だけ，デメリットに感じるかもしれません。

と，やっとわかります。プロトタイプ宣言と関数本体の定義とでは，処理内容が異なりますから，a)かb)がわかった時点で，初めて処理を切り替えられます。

関数の宣言を処理する場合，関数の名前や引数が必要ですから，a)で処理を切り替えたあと，再びポインタを行の先頭に戻して，戻り値の型はvoidで，関数の名前はmainで，引数は……と処理しなければなりません。

このように，せっかく進めたポインタを戻すことをバックトラック(Backtrack)といい，言語処理プログラムでは，このバックトラックが少ないほど，よい「文法」の定義であるとされています。

バックトラックがあると，処理する手順がとても複雑になってしまう場合があります。

たとえば，

```
void
main (
int argc,
char ＊argv[ ]
)
{
```

というスタイルのC言語プログラムでは，‶{″を認識した時点で，5行前の‶void″にポインタを戻さなければなりません。この何行戻す，という作業は，コンピュータのあまり得意な分野ではなく，最悪，ディスクアクセスを1回余計に行わなければなりません。これは，何ナノ秒単位で動いているPCにとって，とてもとても時間のかかる処理です。

これを避けるために，関数のプロトタイプ宣言は，Visual BASICのように，

```
declare void main (int argc, char ＊argv[ ]) ;
```

と，必ず‶declare″で始める決まりを作れば，言語処理系プログラムの作成は楽になります。‶declare″を読み取った時点で，関数の宣言の処理を始められますので，バックトラックが発生しません。

言語の文法という意味では，こちらのほうがよいのですが，コンピュータで扱いやすい文法が，人間にとって扱いやすいとは限らないわけで，結局，こんな宣言方法は，C言語にはありませんが……。

式の評価（四則演算）

たとえば，

1＋2×3 …………… (1)

の値は7です。これは，加算・減算より先に乗算・除算を行う，というルールがあるため，上式は，

(1＋2)×3 ……… (1-a)

ではなくて，

1＋(2×3) ……… (1-b)

と解釈されるからです。リスト1のプログラムでは，入力された数値を左か

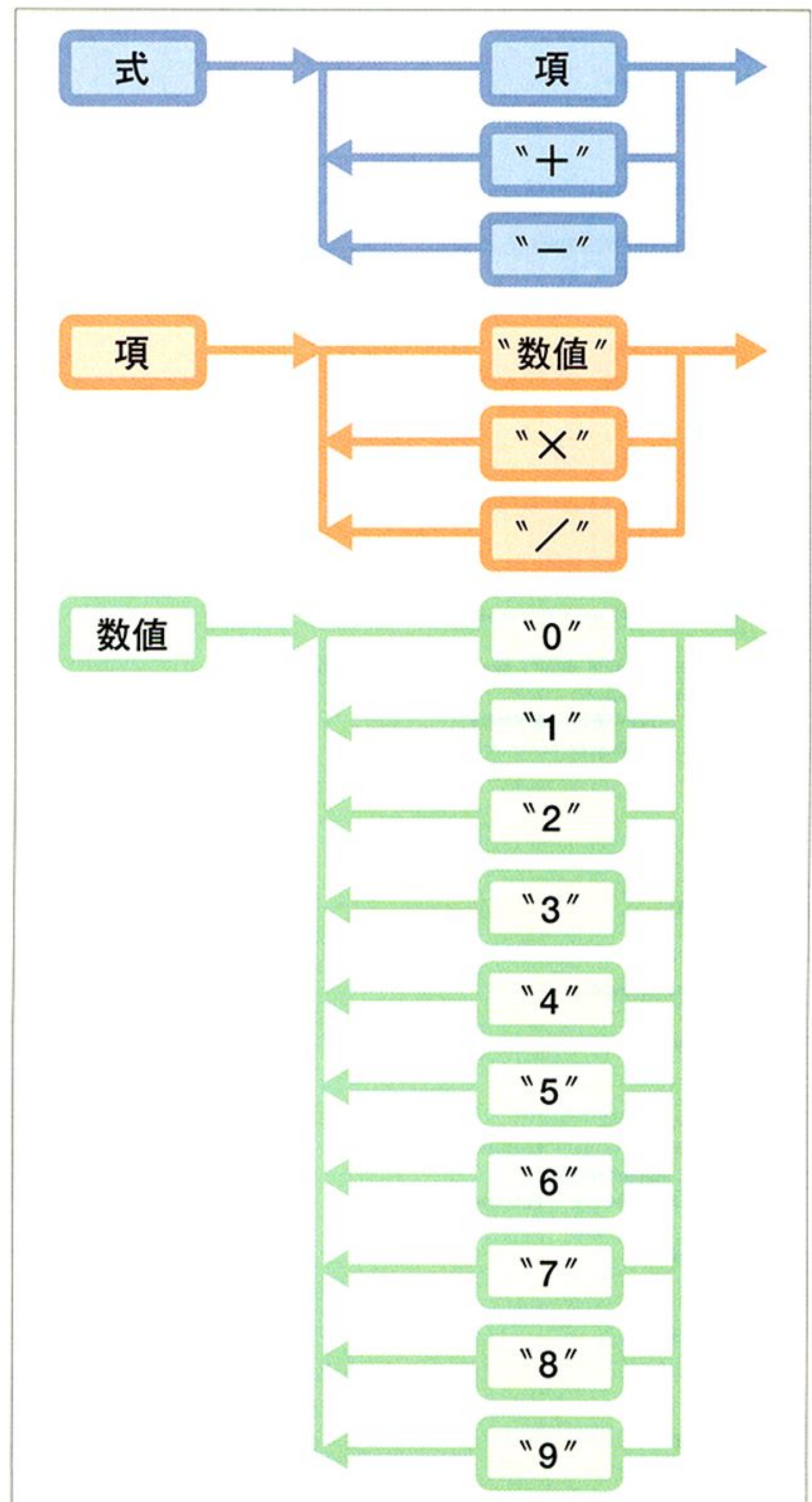



図1
四則演算式の定義

ら順に処理するだけでしたから，単純に演算子の種類を増やすだけの拡張を行うと，式(1)を式(1-a)と解釈してしまいます。そのため，加算を行う前に，右側に乗算が残っていないことを確認する必要があります。処理順番を右からにしても同じ事で，式(1)は正しく式(1-b)として，処理できるようになりますが，逆に，

　1×2＋3　……………（2）

を，

　1×(2＋3)　………（2-a）

と認識してしまい，正しく処理できません。この場合，加算の左側に乗算命令が残っていないかを確認する必要があります。

式(1)と式(2)の両方を扱えるためには，「加減算を行う前には，左右の乗除算が実行されるまで待つこと」というルールをプログラムしなければなりません。

このルールの実現方式には，さまざまな方法が考案されていますが，私がもっとも効果的だと思っている一例を紹介します。この方法は，式を単に扱えるだけでなく，式の文法チェックを単純に実現できるという利点も持っています*3。

簡単のため，加減算を"＋"，乗除算を"×"という記号で表すと，

　＜数値＞×＜数値＞＋＜数値＞×＜数値＞
　（A：＜数値＞×＜数値＞，B：＜数値＞×＜数値＞）

という形を＜式＞の基本形とします。

ここで，式(1)は，Aの部分を＜数値＞に置き換えたもの，式(2)の部分を＜数値＞に置き換えたもの，という解釈をすれば，両方を処理できます。このためには，Aは，"＜数値＞"か"＜数値＞×＜数値＞"どちらでも，対応できなければなりません。以後，便宜的にAやBの部分を項(term)と呼ぶことにします。

リスト2　四則演算をサポートした電卓

```
//
// 四則演算の処理を行うプログラム
//

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

void   main(void);
double ProcessExpression(void);
double ProcessTerm(void);
double ProcessNumerics(void);
int        Amatch(char *);

char   *tokenPtr;      /* トークン分析のためのポインタ */
char   inputBuff[300];

void
main(void)
{
   while(1) {
       printf("Expression> ");
       gets(inputBuff);
       tokenPtr = inputBuff;
       // 何も入力されなければ、終了
       if(tokenPtr[0] == '\0') break;

       printf("Answer> %f\n", ProcessExpression());
       if(tokenPtr[0]) {
           // <式>の後ろに、文字がある
           printf("Syntax Error\n");
       }
   }
}

/*
** <式>の評価
*/
double
ProcessExpression(void)
{
   double value = ProcessTerm(); /* <項>の取得 */

   while(1) {
       if(Amatch("+")) {
           value = value + ProcessTerm();
       } else if(Amatch("-")) {
           value = value - ProcessTerm();
       } else {
           break;
       }
   }
   return(value);
}

/*
** <項>の評価
*/
double
ProcessTerm(void)
{
   double value = ProcessNumerics(); /* <数値>の取得 */

   while(1) {
       if(Amatch("*")) {
           value = value * ProcessNumerics();
       } else if(Amatch("/")) {
           value = value / ProcessNumerics();
       } else {
           break;
       }
   }
   return(value);
}


/*
** <数値>の評価
*/
double
ProcessNumerics(void)
{
   double value;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   if(isdigit(*tokenPtr)) {
       value = atof(tokenPtr);
       /* 読み終わった数字と小数点を飛ばす */
       while(isdigit(*tokenPtr) || *tokenPtr == '.') tokenPtr++;
   } else {
       /* 数字でない */
       value = 0;
       printf("Syntax Error\n");
   }
   return(value);
}

/*
** 与えられた文字列を切り取れれば、文字列文tokenPtrを進め、 1を返す
** 切り取れなければ、tokenPtrを進めずに、0を返す
*/
int
Amatch(char *s)
{
   char    *saveToken = tokenPtr;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   while(*s++ == *tokenPtr++) {
       if(*s == '\0') {
           /* 比較する文字列の最後まで達した */
           return(1);
       }
   }
   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(0);
}
```

<項>＝<数値> | <数値><乗除算><数値>

実際には,

1+2×3+4×5×6

のような式もあるので,

<項>＝<数値> | <数値>{<乗除算><数値>}

あるいは，省略記号([…])を用いて,

<項>＝<数値>[{<乗除算><数値>}]

と表すことも出来ます。

【定義2】

<式>＝<項>[{<加減算> <項>}]
<加減算>＝<"+" | "−">
<項>＝<数値> [{<乗除算> <数値>}]
<乗除算>＝<"×" | "／">
<数値>＝{ 0|1|2|3|4|5|6|7|8|9 }

これで，四則演算のすべて(加算・減算・乗算・除算)が扱えるようになります(リスト2)。世の中には，文法の定義を図示する方法もあって，たとえば，定義2は，図1のように表すことも可能です。これは，開始の点から，すごろくのようにさまざまな経路を経て，式の点にたどりつけば「あがり」です。

＊3 たとえば，1＋2＋＋3は，式として意味をなさないので，文法エラーにすべきですが，処理方法によっては，「加算演算子が連続した場合は，文法エラーとして扱う」と明示的にプログラムしなければ，エラーを検出できない処理法もあります。

括弧の処理

括弧( )がある場合，乗除算のあとに加減算を行うというルールは，順序を入れ代えなければなりません。

(1＋2)×3 …………… (4)

この場合，加算，乗算の順に演算を行います。

先ほど，<式>の部分集合として，<項>を定義したように，ここでは括弧を定義するのに<要素>(factor)という部分集合を定義します。

【定義3】

<式>＝<項> [{<加減算> ＋ <項> }]
<加減算>＝<"+" | "−">
<項>＝<要素> [{<乗除算> <要素>}]
<乗除算>＝<"×" | "／">
<要素>＝" ("<式>") " | <数値>
<数値>＝{ 0|1|2|3|4|5|6|7|8|9 }

上の定義では，<要素>を定義するのに<式>を使っています。当然ですが，<式>を定義するのにも<要素>を使っています。というわけで，この定義は，再帰的(recursive)な作りになっています。

再帰的な定義をプログラムする際には，いろいろなコツがあるのですが，例えば，

(1＋2)×3
b
a

で，式(a)を処理中に，括弧内の式(b)を処理する場合を考えると,

1) 式(a)と式(b)は，同じプログラムで処理する
2) 式(a)の処理(3×3)と式(b)の処理(1＋2)には，異なった値を収める変数が必要

を両立しなければなりません。呼ばれるときによって，扱う値を格納できるよう，<式>，<項>，<要素>を処理する関数で，変数はすべてローカル変数としています(リスト3中，double型変数 value)。

関数の処理

括弧は，優先度を変えるためだけに使われるのではありません。関数の引数を指定する場合も使用されます。リスト3では,

sqrt　√
log　自然対数
sin　サイン(正弦)
cos　コサイン(余弦)
tan　タンジェント(接弦)

を実装してみました。

リスト3を実行し,

Epxression > sqrt (2)

と入力すると,

Answer > 1.414213

と，どこかで見覚えのある数値が表示されます。

単純な括弧の処理に比べて，値を返す前に，それぞれの演算を行っているだけだということがわかると思います。文法的に見ると,

1×(2＋3)

の括弧は与えられた引数をそのまま返す関数，ともいえますね。

その他の演算子

これで，ひととおりの「式」を処理できるようになったのですが，実際のプログラミング言語には，論理演算(andとかorとか)や除算(割り算の余り)や等式·不等式などがあり，もう少し，プログラムの拡張が必要です。

演算子の種類を追加する場合は，その優先度を考慮する必要があります。たとえば,

1＋2×3

で，2×3を先に行うのは，×の方が＋よりも優先度が高いということです。

1＋2×3×(4＋5)

で，4＋5を先に行うのは，括弧が×よりもさらに高い優先度を持っているからです。

演算子の追加は，<項>の追加と同じ要領で行えます。優先度の高い演算子は，数値の処理に近いところで処理し，低い演算子は，式全体の処理に近いところで処理します。

参考までに，C言語では，演算子の優先度は，表2のように定義されています。このように15段階も優先度があると，いちいち<項>とか<要素>とか，それぞれの段階ごとに違った日本語を考えてもいられないので，Expression1とかExpression2のような名前にするとよいかもしれません。

また，実際の処理系では，演算の精度や型も考慮しなければいけないでしょう。たとえば，整数·自然数·文字列をサポートした言語処理系の場合,

整数＋整数

は，整数同士の加算命令で処理できますが,

整数＋自然数

は，一度，左の整数を自然数として扱えるよう型変換(Casting)して，自然

表2　C言語の演算子優先度(抜粋)

優先度	演算子
	( ), [ ], ->, .
高	!, ~, ++, --
↑	*, /, %
	+, -
	<<, >>
	<, <=, >, >=
	==, !=
	&
	^
	¦
	&&
	¦¦
↓	?:
低	=, +=, -=, …
	,

数同士の加算を行います(たいていのCPUは，整数の加算と自然数の加算で，異なる命令を持っていて，前者のほうが効率的に行え，不必要に後者の加算を行わないようにいっているはずです)。また，

文字列＋文字列

の場合は，数学的な「加算」というよりは，文字列の「連結」を行わなければいけないかもしれませんし，

文字列＋自然数

のような式は，エラーにするべきかもしれません。

この各ステップを紹介していくと，「式の評価」を実現するアルゴリズムの骨格がわかりにくくなってしまいますので，今回は扱いません。

この連載の最終回では，これらの実装例として，私家版X-BASICを紹介する予定ですが，興味のある方は，各自リスト3を改造して，いろいろな方法を試してください。

次回は変数の管理を説明する予定です。

リスト3　括弧・関数をサポートした電卓

```
//
// 四則演算と括弧の処理を行うプログラム
//

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

void   main(void);
double ProcessExpression(void);
double ProcessTerm(void);
double ProcessFactor(void);
double ProcessNumerics(void);
int        Amatch(char *);

char   *tokenPtr;      /* トークン分析のためのポインタ */
char   inputBuff[300];

void
main(void)
{
   while(1) {
       printf("Expression> ");
       gets(inputBuff);
       tokenPtr = inputBuff;
       // 何も入力されなければ、終了
       if(tokenPtr[0] == '\0') break;

       printf("Answer> %f\n", ProcessExpression());
       if(tokenPtr[0]) {
           // <式>の後ろに、文字がある
           printf("Syntax Error\n");
       }
   }
}

/*
** <式>の評価
*/
double
ProcessExpression(void)
{
   double value = ProcessTerm(); /* <項>の収得 */

   while(1) {
       if(Amatch("+")) {
           value = value + ProcessTerm();
       } else if(Amatch("-")) {
           value = value - ProcessTerm();
       } else {
           break;
       }
   }
   return(value);
}

/*
** <項>の評価
*/
double
ProcessTerm(void)
{
   double value = ProcessFactor();   /* <項>の収得 */

   while(1) {
       if(Amatch("*")) {
           value = value * ProcessFactor();
       } else if(Amatch("/")) {
           value = value / ProcessFactor();
       } else {
           break;
       }
   }
   return(value);
}

/*
** <要素>の評価
*/
double
ProcessFactor(void)
{
   double value;

   if(Amatch("(")) {
       value = ProcessExpression();
       if(! Amatch(")")) {
           /* 括弧が正しく対応していない */
           printf("Syntax Error");
       }
   } else {
       value = ProcessNumerics();
   }
   return(value);
}

/*
** <数値>の評価
*/
double
ProcessNumerics(void)
{
   double value;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   if(isdigit(*tokenPtr)) {
       value = atof(tokenPtr);
       /* 読み終わった数字と小数点を飛ばす */
       while(isdigit(*tokenPtr) || *tokenPtr == '.') tokenPtr++;
   } else {
       /* 数字でない */
       value = 0;
       printf("Syntax Error\n");
   }
   return(value);
}

/*
** 与えられた文字列を切り取れれば、文字列文tokenPtrを進め、 1を返す
** 切り取れなければ、tokenPtrを進めずに、0を返す
*/
int
Amatch(char *s)
{
   char    *saveToken = tokenPtr;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   while(*s++ == *tokenPtr++) {
       if(*s == '\0') {
           /* 比較する文字列の最後まで達した */
           return(1);
       }
   }
   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(0);
}
```

Oh!X 春号
読者モニター/プレゼント

Visual Studio 6用分散アプリケーションの設計/開発のマスター教材
「Mastering Distributed Application Design and Development Using Microsoft Visual Studio 6」日本語版
提供：マイクロソフト
☎03-5454-2300　☎06-6245-6995

C 2名

A 1名

クロスプラットフォーム対応
総合開発環境
CodeWarrior Professional 4
提供：メトロワークス/ビー・ユー・ジー
☎03-3486-6710

SIIのウェアラブルPC「Ruputer」対応開発環境「CodeWarrior IDE for Ruputer」発売記念
Ruputer PRO(本体)
提供：メトロワークス/ビー・ユー・ジー

B 1名

■モニター応募要項
■応募資格：対象となる分野でプログラム開発の意欲のある方。本誌綴じ込みの愛読者カードもしくは官製ハガキに必要事項(住所，郵便番号，氏名，年齢，職業，連絡先，モニターの希望番号と商品名)を明記のうえ，ご応募ください。応募の締め切りは'99年6月1日(当日消印有効)です。
■当選者の発表は発送をもって代えさせていただきます。

1 2名

ノートPCプログラマ必携の
USBテンキーパッド
TK-LUSM
提供：エレコム
☎03-3981-4491

■プレゼント応募要項
■本誌綴じ込みの愛読者カードに必要事項(住所，郵便番号，氏名，年齢，職業，連絡先，プレゼントの希望番号と商品名)を明記のうえ，ご応募ください。応募の締め切りは'99年6月1日(当日消印有効)です。
■当選者の発表は発送をもって代えさせていただきます。

注)記入もれ，および希望商品が複数記入されている場合は無効にさせていただきますので，ご了承ください。また，商品の都合上，希望されたものと違う商品が当選する場合があります。
注)雑誌公正競争規約の定めにより，プレゼント当選者は本号のほかのプレゼントには当選できない場合があります。

3 3名

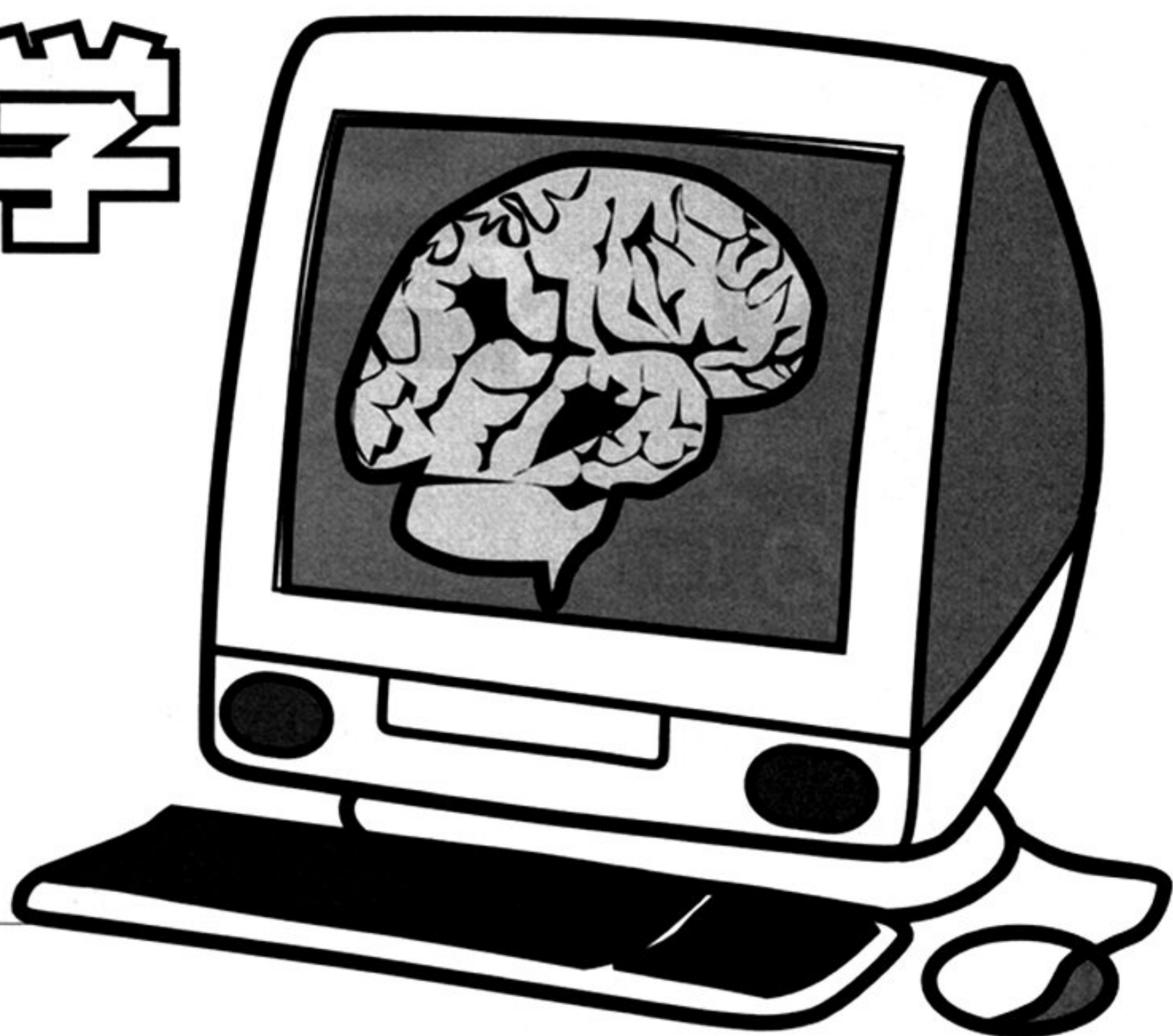
復刊号でプレゼントしたスタジャンと同じ刺繍を使ったOh!X印のオリジナルキャップ
Oh!Xオリジナル帽子
提供：「くらぶ・68」

2 5名

アメリカンな香りのピンボールポスターと一般には入手できないピンボール台のバックグラスをデザイナーのサイン入り(一部)で
Williams社各種ピンボールポスター&バックグラス
提供：マインドウェア
※ポスターの種類などは指定できません

拡張電脳学入門講座

清く正しく21世紀

祝 一平 Iwai Ippei

まずは，キリスト教関係の方々に苦言を呈したい。キリスト教の暦＝西暦では，どうやら「西暦0年」がないらしい。つまり，キリストが生まれた年を「A.D.1年」とし，その前の年は「B.C.1年」としているのだ。

そーゆーことをするから西暦2000年がまだ20世紀で，西暦2001年にやっと21世紀になるという，オシリがムズムズするようなことになるのだ。まあ，「0」を発見したのはインド人だそうだから，しかたがないともいえるのだが，とにかく今後はそのようなことがないように万全の注意を払っていただきたい。

<いきなり閑話>

現在，私は在野の哲学者（ウププッ）として，日夜どーでもよいことに思索を巡らせている。ここで肝心なのは，役に立たないことに専念するという努力である。さもないと思考の純粋性という美しさを保てなくなるからなのだ。

さて最近の私の研究は『否定歴史学』，『計量主観学』，『逆進化人類学』などである（なんなら，もっとあるゾ）。

まずは，『否定歴史学』である。これは私が大学時代に聞いた西洋史の講義が発端となっている。その日の講義のテーマはカルト的なことに「魔女」であった。その講義では魔女の判別方法などの有益な事柄を教わったのであるが，教授は明快な論理によって，時の権力者（王）たちは魔女が実在しないことを認識しており，（実際には存在しない）魔女を政治の道具として使っていたという解説をしてくれた。

その論理とは「魔女を利用することができれば，それは絶大な戦闘力であるから，必ずや『ほうきに乗った夜間空軍』を結成したはずだ」，ということである。簡略化すると，

魔女軍団は結成されようとしなかった

↓

権力者（王）は魔女が実在しないと知っていたなのだ。

私はこの教授から得たこの論理システムを基として，『否定歴史学』の研究に勤しんでいる。で，最近の成果は「ヤマタイコクはどこにあったか」という，いささか赤面してしまうテーマについてである。

もしも同じ説を唱えている歴史家（エセも含む）がすでにいるならば，一笑に付していただきたい。少なくともここに述べた説は，私が独立に結論したものである。

まあ，結論からいってしまうと，「魏志倭人伝に記載されているヤマタイコクの位置は，デタラメである可能性が高い」ということである。

なぜか？

それはなぜかと問われるならば，教えてやるのが世の情け。つーわけで，簡単に解説すると，ヤマタイコクの使者がはたして「本当のことをいうだろうか？」である。だって，もしも魏がマジで攻め込んできたらどーすんだ？

魏にはそんな気はまったくなかったようだが，第三国はどう思うかわからない。つまり，自国の地理的位置を明らかにすることは，国防上非常に危険なことなのだ。

よって私は結論するのである。

「ヤマタイコクの位置は，魏志倭人伝に伝えられるルートから大きく離れたところにある」

◆ ◆ ◆ ◆ ◆

次に『計量主観学』であるが，これは「誰もが知っていること」を基準として，さまざまな事柄を数値化しようという試みである。まあ，遊びとしても面白いので，お勧めの一品だったりする。

具体的には，最初はパンチョ値から始まった。これは，

パンチョ伊藤＝1パンチョ

とし，電車の向かいに座ったオジサン（ああっ，俺もその領域だっ）とか，テレビに出てきたオジサンとかのパンチョ値を「断定」するのである。

たとえば，テレビの番組だけでなく，CMにもよく出てくる「神田川の飯屋のオヤジ」は，0.9パンチョである。あと，数例を挙げると，突然辞めてしまった首相は1.8パンチョ（芸術的なまでによくできてるねぇ），山城シンゴ＝0.5脳天パンチョ，などである。

まあ，一般的に，前髪の生え際が見えないとか，昔から髪型が全然変わらないとかだとパンチョの可能性があるので，これをチェックすればテレビを見る楽しみも倍加しようというものだ。

以上が，私のパンチョ値の解説であるが，私の研究によれば，5パンチョという過激な芸能人がいる。ここでは名前を明らかにはしないが，奴は下駄を鳴らしてやってくるらしい（おおっとぉ）。

<閑話休題>

以前，私は「インテル係数」という指標を提案したことがある。これは，インテルが発表した性能値を誰も再現できないという事実に由来する。おそらくはインテルの本社のどこかに，キャッシュをシコタマ載せたマシンがあって，ギンギンにオプチマイズされたコードを出すコンパイラ(1回コンパイルするのに半日以上かかる)もあって，それを使ってベンチマークを出していると邪推した次第である。

このインテル係数の発見を始点として，『拡張電脳学』が始まる。これは『空想電脳学』と呼んでもいいのであるが，SF的な雰囲気がイヤなので〝拡張〟としている。

まずは「VA(バーチャルアコースティック)音源」の拡張である。VA音源はまだ「本物に近づけるための手法」にすぎないが，おっとどっこい，こいつをスーパーコンピュータ並べてマジにやったらどうなるんだい？

どーなるもこーなるも，ストラディバリウスの音響的特性(パラメータ)を与えてやれば，スーパーコンピュータはストラディバリウスの(デジタイズ合成された)音を出すのだ。

さらに拡張させるならば，オーケストラの各楽器も，名器をもとにパラメータを与えてやる。すると，バイオリン，ビオラ，チェロ(これは少し疑問)がぜーんぶストラディバリウスである。

さらには，木管，金管，ピアノなどを全部パラメータにしてスーパーコンピュータに叩き込むと，「デジタルオーケストラ」が完成する。こうなると，指揮者のやることはスーパーコンピュータにマクロなパラメータを与えることだけになる可能性がある。

いや，なる。

なぜならば，まず「スーパーストラディバリウス」の可能性があるからだ。つまり，いまだかつて人類が作りえなかった超名器をスーパーコンピュータの中に構築しうるのだ。

さらには，一流の演奏者になるには，天賦の才能と何年ものキツいレッスンが必要であるが，SVA(スーパーVA)音源では，ポンとパラメータを与えるだけである。こうなると演奏会に出かけるというのは懐古的な趣味になるといわざるをえなくなる。さて，ヨーヨー・マの運命や如何に？

◇　◇　◇　◇　◇

音楽の面でこのようなことが起これば，当然映像でもビビンバである。

この原稿を書いている現在，テレビで「バグズライフ」の映像をよく見るが，これなどもまだまだ「よくできたCGアニメ」の領域を脱していない。

そこで私は予言する。

マリリン・モンロー，ナウ リターン！

まずはだな，技術的エントリーとしておそらくこうなるというのが，

「マリリン・モンロー(以下MM)に体型と所作が似た女優を探し出し，映画を作り，顔にMMの表情をマッピングする。声もMMのモノ真似で吹き替える」

である。ここで大事なのは，モンロー・ウォーク(お尻を振るやつね)もキチンと再現することである。

これを無理だと思うか？

そのように思う方には，映画『アポロ13』の発射シーン(バリバリの合成)を見たNASAのスタッフが，「こんな映像がどこにあったんだ！？」といったという事実をお伝えしておこう。昨今のCGのクオリティを考えれば決して大げさな話でもない。

私が愚考するに当たって，現在もっとも「復活」が求められている俳優はMMである。だから，まもなくバーチャル・マリリンの時代なのだ。

以上に「拡張電脳学」の成果の一部を紹介させていただいた。これには，ぜひとも読者の方々に参加していただきたい。なーに，要領は簡単である。コンピュータをネタにして，もっともらしいホラを吹けばよいのだ。ケケケ。

<さてさて>

「真っ黒ソフト」とかゆーOSソフト会社がノシている現在，我々に未来はあるのだろうか？

ある。

横になってハナクソをほじっていても時間は勝手に流れていく。働きたい人はセッセコと働き，ちゃんと我々を21世紀に運んでくれる。絶対にいつかはバーチャル・マリリンが出現するだろうし，きっとガンは怖い病気ではなくなる。

実をいうと，私の机の上には，おむすびコロリンでiMacが載っているのだが，それを「20.99世紀のもの」といったら笑われるだろうか？う〜ん，やっぱり笑われるかもしれない。でも真っ黒ソフトの産物は，どう考えてもベタで20世紀の恥だ。

2001年まであと1年半余りである。在野の哲学者として，これからも私は，どーでもいーよーなことを考え続けるだろう。

それでは，とりあえず1999年の7月が無事に済むように願いつつ，私はコタツの中にもぞもぞと潜り込むとしよう。では。

illustration：Yamada Haruhisa

知能機械概論

Tinoukikai-Gairon

第101回

Papert博士からの2つの贈りもの StarLOGOとMINDSTORMS

有田隆也 Arita Takaya

ごみ集めの謎

次のような課題が皆さんに与えられたとしましょう。

「床にごみがたくさん落ちている。それを多数のエージェントに拾わせてかき集める」というプログラムを作れ

エージェントという言葉はいろいろな使われ方がされますが，ここでは動き回るロボットをシミュレートする自律的に振る舞う個体を記述したソフトウェアとでもいっておきましょうか。エージェントという言葉を使わなくても，平面上にものがたくさん落ちていて，それをたくさんのロボットに拾ってかき集めさせるというシミュレーションのプログラムを作れというようにいい換えても，意味は伝わるでしょう。

ただし，そのロボットというのが，備えているセンサがきわめて貧弱で，周りのごみをきょろきょろ眺め回すことはできなくて，ごみのあるところに行ってはじめてその存在がわかるものとします。また，各ロボットはコミュニケーション機能も貧弱で，お互い同士自由に通信できないとします。さて，どのようなアルゴリズムが読者の皆さんの頭に浮かびますか？

さっそく，ここでおそらく皆さんがあまり知らないようなプログラミング言語で書かれたプログラムをひとつお見せしましょう（リスト1）。

どうでしょうか？　たぶん，この記事を読んでいる読者の方のプログラミングに関する熟練度が高ければ高いほど「こんな短いプログラムでよくも～」と驚いたのではないでしょうか。これですべてですから。でも，驚くべきは，プログラム量というよりは，そのアルゴリズムにあるといえます。

ごみを一面にランダムにばらまいたあとに，エージェントたちの仕事は始まりますが，このプログラムで表現されているアルゴリズムは，次のようなものです。

普段はランダムに動く
もし，ごみ山に踏み込んだ場合，
**　手ぶらなら，**
**　　ごみ山からごみをひとつ取る**
**　すでにごみを持っているならば，**
**　　ごみ山に置く**

というなんともシンプルさのきわまったアルゴリズムです。これに従ってすべてのエージェントを平面上で動かします。このアルゴリズムを見て，すぐに「なるほど，これはうまくいくな」とピンとくるならば，あなたはすごすぎると思います。

普通，このような課題が与えられると，どうやってゴミを効率よく探したらいいだろうかとか，拾ったらどこに持って行ってごみの山を作ればいいだろうかなどと，アルゴリズムを考えていくのが常道だと思うのです。でも，このアルゴリズムは，そういう手法とは大きく異なった考え方に基づいています。

さっそく，種明かししてしまいますけれど，書かれているとおり，このアルゴリズムに従うエージェントたちは，なにも考えないでふらふらとそこらをさ迷います。そして，ごみ山に乗り上げると，そのときごみを持っているならば，ごみをそこに置きます。これは，まあ，ごみ捨て場にごみを捨てるのですから，悪くないですよね。問題は，ごみを持っていないときですが，そのときは，必ずごみ山からごみを拾っちゃうのです。

このごみ拾い行為は問題ありそうですよね。だって，せっかく集まったごみ山でも，それを崩してしまおうということですから。でも，ごみ山のごみの個数が少しならば，これは拾うことが必要です。だって，ごみをかき集める＝ごみ山を作る＝小さいごみ山からごみを持ってくる，ということですから。

大きな山からごみを拾うことは，せっかく集まってきた大きなごみ山を切り崩してしまうことに相当します。逆に，小さなごみ山からごみを拾うことは，小さなごみ山のごみを大きなごみ山に移動するために必要なことです。しかし，問題はここですが，そのどちらになるかは，まるで運任せなのです。

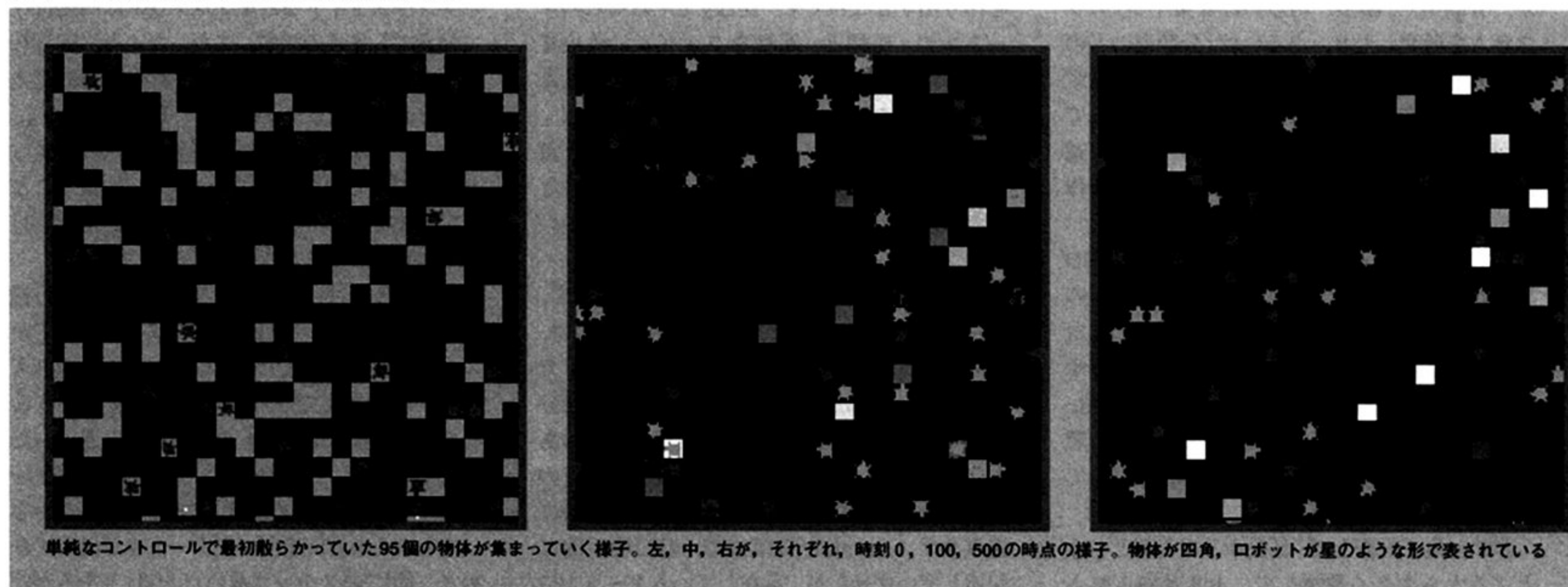
リスト1　ごみ集めプログラム

```
patches-own [garbage]
turtles-own [ahead inhand]

to setup
ca
ifelse (random 100) < density [setgarbage 1][setgarbage 0]
if garbage = 1 [setpc green]
crt number
setc green
setxy random screen-size random screen-size
setinhand 0
settime 0
end

to go
check
rt random 60
lt random 60
fd 1
scale-pc green garbage 0.1 10
end

to check
ifelse (inhand = 0) and (garbage > 0)
[setinhand 1 tsetgarbage garbage - 1 setc red]
[if (inhand = 1) and (garbage > 0)
[setinhand 0 tsetgarbage garbage + 1 setc green]
]
end
```

図1　ごみ拾いプログラムの実行画面

単純なコントロールで最初散らかっていた95個の物体が集まっていく様子。左，中，右が，それぞれ，時刻0，100，500の時点の様子。物体が四角，ロボットが星のような形で表されている

実は「ごみ山があればそこに置く」というほうも，必ずしもよいとはいえないことに気づきます。だって，その山が1個のごみだけで作られているとするならば，そこに置くことはよくないことが多いですよね(最初は全部の山は一個のごみでできているから，いつもだめということでは，永遠にごみは集まらないけれど)。

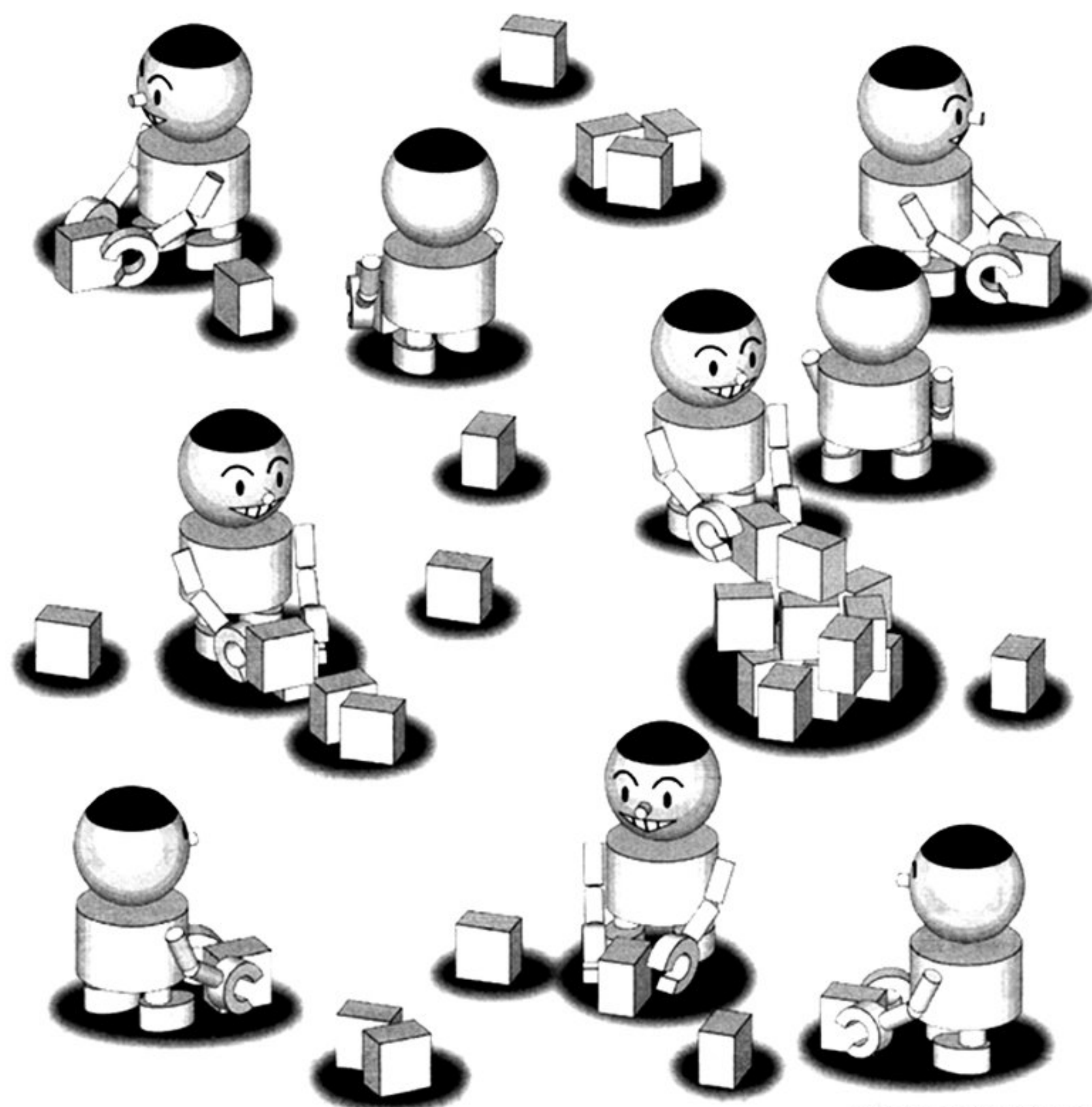
そろそろ，このアルゴリズムの意味が明らかになってきたと思います。プログラムを実行させて，画面上でごみ山がどのようになるのかを眺めているとさらによくわかるのですが(**図1**)，ごみ山にごみが積まれたり，ごみ山からごみが運び出されたりということが，無秩序に繰り返されますが，重要なことは，「一度，消えたごみ山は決して戻らない」ということです。一面にごみ山があります。それらの山は高くなったり，低くなったりを繰り返しますが，数としては，なくなる一方なのです。そういうわけで，各エージェントたちは，まるでなにも考えずに単純な作業に専念しているのですが，全体としてはお片づけは着々と進む(はず)というわけです。

Illustration：Yamada Haruhisa

LOGOからStarLOGOへ

先ほどのプログラムはStarLOGOというプログラム言語で書かれています(現在はMacintosh用のみがリリース。Windows版はα版作成が進行中)。StarLOGOは，LOGOという20年以上前に作られた言語を基にしています。LOGOは，人工知能の研究領域の基盤を作ったひとりであるS.Papertが教育目的のために作った言語で，基本的にはLisp言語のようなリスト処理言語に属し，データの型宣言が不要な言語です(15年前にソフトバンクのアルバイトでMZ-LOGOのマニュアルを書いたのが懐かしい)。

LOGO言語は，とても馴染みやすい言語で，定義されたコマンド名をキーボードから入力することによって，会話的にカーソル(タートル，亀と呼ぶ)を画面上で動かして容易に線画を描くことができます。抽象的に計算をさせるより，亀さんに「右向け」とか「まっすぐ10ステップ進め」などと命令を打ち込むと亀が画面上でその命令に従って動くほうが，子供たちがやる気になるのはいうまでもありません。亀が動いた軌跡に線を描くことができます。こういう要領でお絵かきをすることをタートルグラフィックと呼びます。簡単な手続きを自分で定義してコマンドにすることにより，容易に複雑なプログラムを作っていくことができるのが特徴です。

一方，StarLOGOはPapertの弟子のResnickが設計した言語［1］で，LOGOとは雰囲

気がかなり異なります。それは，StarLOGOが，分散型（分権型，あるいは非集中型）のシステムを研究したり，あるいは「分散型のものの見方」とでもいうべきものを身につけたりするためのモデリング環境を目指しているからです（研究用といっても機能的に非力な点や，速度が遅い点があるので，大きなプログラムを作る前のプロトタイプ的に使うのがベスト）。

そのため，次の2つの大きな拡張をLOGOに対して行っています。ひとつは，並列実行の動作を自然に記述することができるようにしたこと，つまり，多数のタートルを同時に動かすことができるようにしたこと，もうひとつは，より複雑なモデリングを可能とするために，環境というものを実体化して，その動作に関しても記述できるようにしたことです。

単に，亀を画面上で1匹動かしてみるというところから大きく飛躍して，セルオートマトン，生態系，物理・化学現象，経済，交通渋滞などのプログラミングが，自由に，しかも通常のプログラミング言語に比べて，ずっと簡単にできるようになっています。

無論，得意/不得意はあります。先に述べたように，なにか中央に全体を統率するようなメカニズムがあって，それに従って全体が動いているというのではなく，分散化されたユニット間の相互作用から秩序あるパターンが生まれるような分散システムの基本的なシミュレーションにとても向いています。

プログラミングをする際，3種類の主体を対象とすることになります。それは，タートル，パッチ，観測者です。タートルは基本的にはLOGOにおけるタートルと同様ですけれど，LOGOにおいては軌跡を描くものでしたが，StarLOGOではそれだけでなく，さまざまな振舞いをします。パッチは環境に相当し，タートルがその上で行動する環境を構成するものでして，マス目のように区画分けされています。観測者というのはより一段高いレベルからこの小世界を観測するものです。統計を取ったりします。

プログラムの正体

では，先のプログラムを簡単に説明することにしましょう。冒頭のブロック（2行）はパッチとタートルで使われるローカル変数をそれぞれ宣言しています。2つめのブロックがsetup手続きを定義する部分でして，最初のほうで，ごみが落ちているパッチを乱数で決めています。densityはこのプログラムの外で与えられるパラメータです。このプログラムのユーザーインタフェイスのところで，簡単にボタンなどを設定して，プログラムの実行前に設定するようにします。

```
crt number
```

というところが肝心のタートル（エージェント）を作り出すところです。setxyでそれぞれのタートルのx座標とy座標をランダムに決めます。setxyとひとつしかコマンドは書きませんが，すべてのタートルで同時に実行されます。3番目のブロックが本体でして，普通goと書かれたボタンを定義して，そのボタンを押すとこの部分が実行されるように関連付ければいいのです。

中身は次のようになっています。まず，checkというコマンドを実行します。これは，4番目のブロックで定義されているコマンドです。ここでは，ごみの載っているパッチに自分がいるならば，処理を行う部分です。そのあと，最大60度，乱数を選んで，右と左にその角度だけ向きを変えます。そのあとで1歩前に進みます。checkからfdまで，特に明示的に記述していませんが，存在しているタートルにおいて同時に実行されるというところが，ミソです。

次のset-pcでは，ごみの数に応じて，パッチの色を塗り分けるというところで，これはすべてのパッチで同時に実行されます。最後のブロックは，タートルがごみの載ったパッチに乗り上げたときに，先に示したアルゴリズムどおりに，拾ったり落としたりする部分です。

StarLOGO言語の基本

StarLOGOプログラムの基本的な構造をごく簡単にまとめておきましょう。｢｣，および，｢｣で挟まれた部分は，説明の記述用です。

```
手続き
to 手続き名
[手続き本体の処理]
end

値渡し/値返し付き手続き
to 手続き名 :引数1 :引数2
[処理]
output 返値
end

if構造
if [条件式]
[[真の場合の処理]]
if else 構造
ifelse 条件式
[[真の場合の処理]] [[偽の場合の処理]]

繰り返し構造
repeat 繰り返し数
[[処理]]
```

ユーザーインタフェイス部分の制作の容易さや，インタラクティブ性も素晴らしいものです。簡単にマウスでちょいちょいとやるだけで，ボタンやスライドなどをプログラムに関連付けて使えてしまうのです。また，なんと，実行している最中に，好きな変数の値を自由に変えることができてしまうのです。それだけでなく，動いているタートルを無理やりマウスでズルっとひきずって別の場所に持ってきてしまうこともできます。もう少し詳しいことは，拙著[2]にも書いていますので，ご覧ください。

あとひとつ付け足しておきます。並列実行言語としたために，タートルグラフィクスもずいぶんと様変わりしました。LOGOでは，100ステップ前に進んでから90度右を向くということを4回繰り返せば，正方形が書けるといった感じでしたが，StarLOGOでは，5000個タートルを作る（crt 5000）と，フィールドの中心にばらばらな方向を向いたタートルが出現するので，前に50ステップ進め（fd 50）とやるだけで，半径50の円が描けてしまうのです。

なんといっても使ってみることがいちばんです。StarLOGO言語は，MIT Media LabのM Resnickが考案し，現在でも同所で開発やユーザーグループの運営などを行っていますが，Tufts大学の研究グループが，Star Logo Tと銘打って新たなバージョンの開発を始めています。これは，StarLOGOをいくつかの点で拡張したものとなっており，もっと使いやすくなっています。両者の処理系とも，ホームページ[3，4]からダウンロードすることができます（両者ともMac用のみ）。多数のサンプルプログラムもついてきますし，ホームページ上に詳しいマニュアルも載っています。

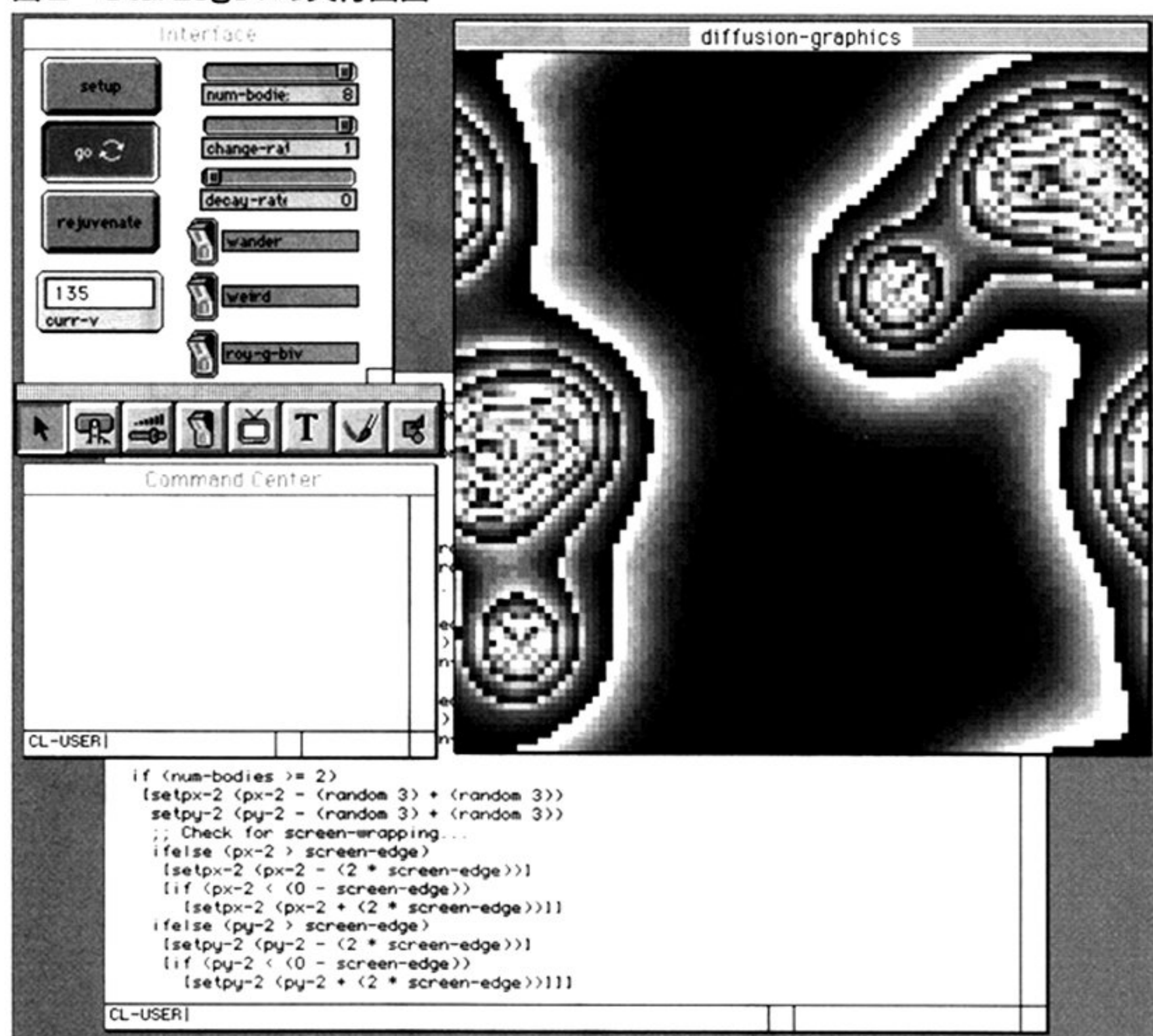
「構築主義」という主張

世の中，流れとしては「分散」に向かっています。インターネットによる情報発信，コンピュータアーキテクチャ，分散人工知能でもそうですし，地方分権もそうでしょう，あるいは，ネットワーク型ゲームもそうですし，とにかくなにからなにまで，分散型一直線という感じです。

しかし，そうはいうものの，我々の頭の中は，やはり従来からの考え方，なにか原因がひとつあってそれが大きく影響しているのだろうとか，冒頭にあげたごみ集めのさせ方でもそうですし，ついつい，ひとつの流れでものごとを考えていきがちです。そこら辺をとても明快に指摘したのが，Resnickの本[1]なのです。彼は，StarLOGOを高校生などに使わせて，分散的なものの考え方を身につけさせようということを主張しています。

さて一方，LOGOを作ったPapertのもともとの主張は，「自分で身の周りのことを題材にして自分の興味に基づいて，作り上げていくことによって学習していくことはとても重要なのだ。そして，その際にコンピュータは素晴らし

図2　StarLogoTの実行画面

```
if (num-bodies >= 2)
 [setpx-2 (px-2 - (random 3) + (random 3))
  setpy-2 (py-2 - (random 3) + (random 3))
  ;; Check for screen-wrapping...
  ifelse (px-2 > screen-edge)
   [setpx-2 (px-2 - (2 * screen-edge))]
   [if (px-2 < (0 - screen-edge))
     [setpx-2 (px-2 + (2 * screen-edge))]]
  ifelse (py-2 > screen-edge)
   [setpy-2 (py-2 - (2 * screen-edge))]
   [if (py-2 < (0 - screen-edge))
     [setpy-2 (py-2 + (2 * screen-edge))]]]
```

いほど役に立つのだ」ということです。この主張をひと言で表すと，“constructionism”(構築主義)です。

構築主義の立場からいうと，コンピュータの中だけで作り上げるよりは，自分の手でなにかものを作り上げたほうがより効果的であることは明らかです。そこで，コンピュータで実際のロボットを制御し，タートルへのコマンドにより，ロボットが指示どおり動くキット(LEGO/Logo)も売り出され，教育の現場でも利用されています。

これは，Papert，Resnickらと LEGO社が共同開発したもので，おもちゃのレゴと同様の部品スタイルでできています。

構築主義と，Resnickの主張する分散的な考え方とは，どのような関係にあるといえるのでしょうか？　自分でプログラムを作って理解していくというところは，構築主義そのものです。ただし，設計者が意図するとおりに作るのではなく，分散されたコンポーネントから，設計者の意図とは離れたところでシステムが創発された挙動を示す現象を観察するというのがStarLOGOですから，その面からは，正反対のアプローチにも一見思われます。でも，よく考えると，StarLOGOでは，プログラマは全体のシステムの挙動を設計するのではなく，各コンポーネントの従うルールだけを設計するのですから，その指摘は当たりません。

しかし，Papertの主張する構築主義の考え方の自然な発展は，やはり，LEGO/Logoであるといえましょう。手でものを作り上げるという点が，StarLOGOからは，残念ながらすっぽり抜け落ちているというところは事実ですから。

さて，博士からの2つの贈りものということで，お待たせしました。Papertの構築主義，あるいはLOGO言語からの延長線上のアプローチとして，StarLOGOだけではなく，もうひとつとても魅力的なモノがあるので，それを次に紹介します。

万人のための永遠のおもちゃ「MINDSTORMS」

「永遠のおもちゃ」，それはLEGOブロックだと僕は断言します。その理由は，壊れないからです。(自分の周りにお行儀のよいお友達ばかりしかいなかった？　からかどうか知りませんが)ブロックが割れているのを見たことが一度もないのです。部品が次第に紛失していってなくなることはありえますが，あのおもちゃこそ，永遠にリサイクルできるエコロジカルなおもちゃだと思います。歯で部品を外した痕跡は増やしながらも。

そして，いま，あのブロックのひとつにコンピュータが入り込み，年代を問わないオール世代向きの最強のおもちゃとなったのです。「永遠かつ万人のおもちゃ」，それが「MINDSTORMS」です。

PapertやResnickらのグループは，構築主義の立場から，Programmable brickというプロジェクトを1970年代半ばから行っており，そのひとつの成果がLEGO社が製品化したMINDSTORMSといえます。

基本セットになにが入っているかというと，まず，8ビットプロセッサの入ったブロック

写真1　MINDSTORMS(基本セットと2つのオプションキット)

RCX(タバコ3箱分くらいの大きさ)，モーター2個，センサ3個(光センサ1，タッチセンサ2)，各種部品(タイヤ，軸，歯車，コード，装飾用部品，普通のブロック)などです。さらに，コンピュータからRCXにプログラムやデータを転送するための赤外線送受信器IR Tower，テンプレートをマウスで組み合わせることによってプログラムを簡単に作ることができるビジュアル言語や作品例などの入ったCD(Windowsマシン用)が入っているのです。

部品総数700以上(だそうです)入っていて，アメリカではだいたい200ドルで買えるというのですから，安いとしかいいようがないのでは。日本でも，限定販売などで，少しずつ売られるようになってきましたが，レゴジャパンは年末までに正式に販売を開始するようです(たぶん，3万～4万円で)。

僕も去年いち早くアメリカに住んでいる姉に頼んで送ってもらい，個人的に楽しんでいます。でも，やはり，人の作品がホームページとかで紹介されているのを見ると，皆さんの発想にはひれ伏すばかりです。LEGO社のデモンストレーションでサッカーをやらせていたのは，ややインチキ臭い(悪い意味ではありませんので)ですが，バーコードリーダ，スキャナ，2足歩行ロボット，遊園地，チューリングマシン……(絶句)。

付属のソフトウェアは，きちんとプログラミングをするということになると，まったく使いものにならないといい切れるのですけれど，そういう人用にはそれなりの開発環境が揃いつつあります。RCX用のコードを吐き出す独自のCもどきのコンパイラもあり，これは，WindowsだけでなくMacやLinuxでも実行できます。さらに，LEGO社がVisualBasicなどから利用できるOCXを公開していますので，それを利用すれば，通常の言語でプログラミングができます。

なお，MINDSTORMS関連は，Oh! MZ時代からの本誌の読者(カナダのお宅の地下室にMZ誌が保存されているそうです)でもあられるJinSato氏の運営する「MindStorms情報

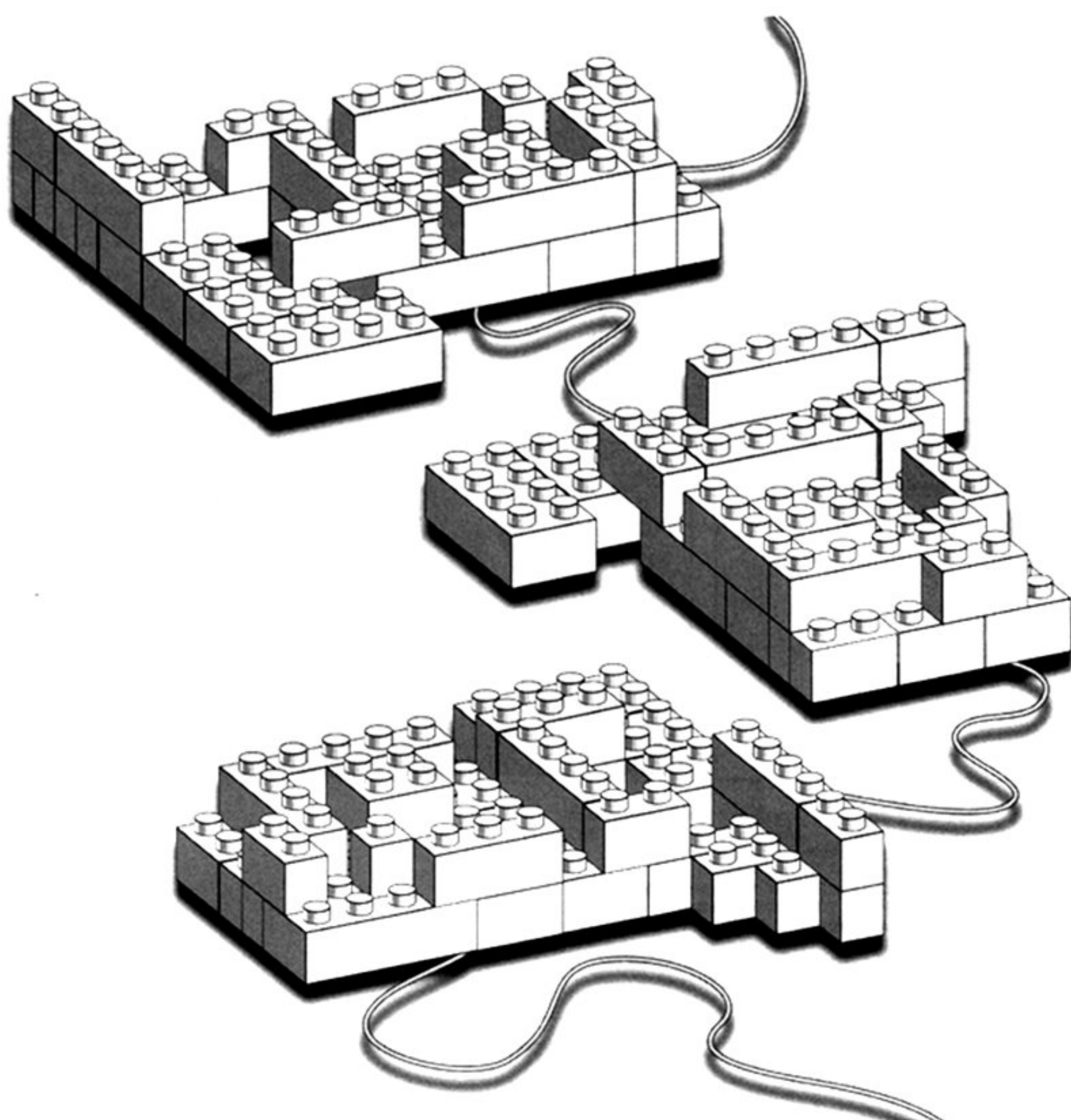



図3　MindStormsの公式ホームページ

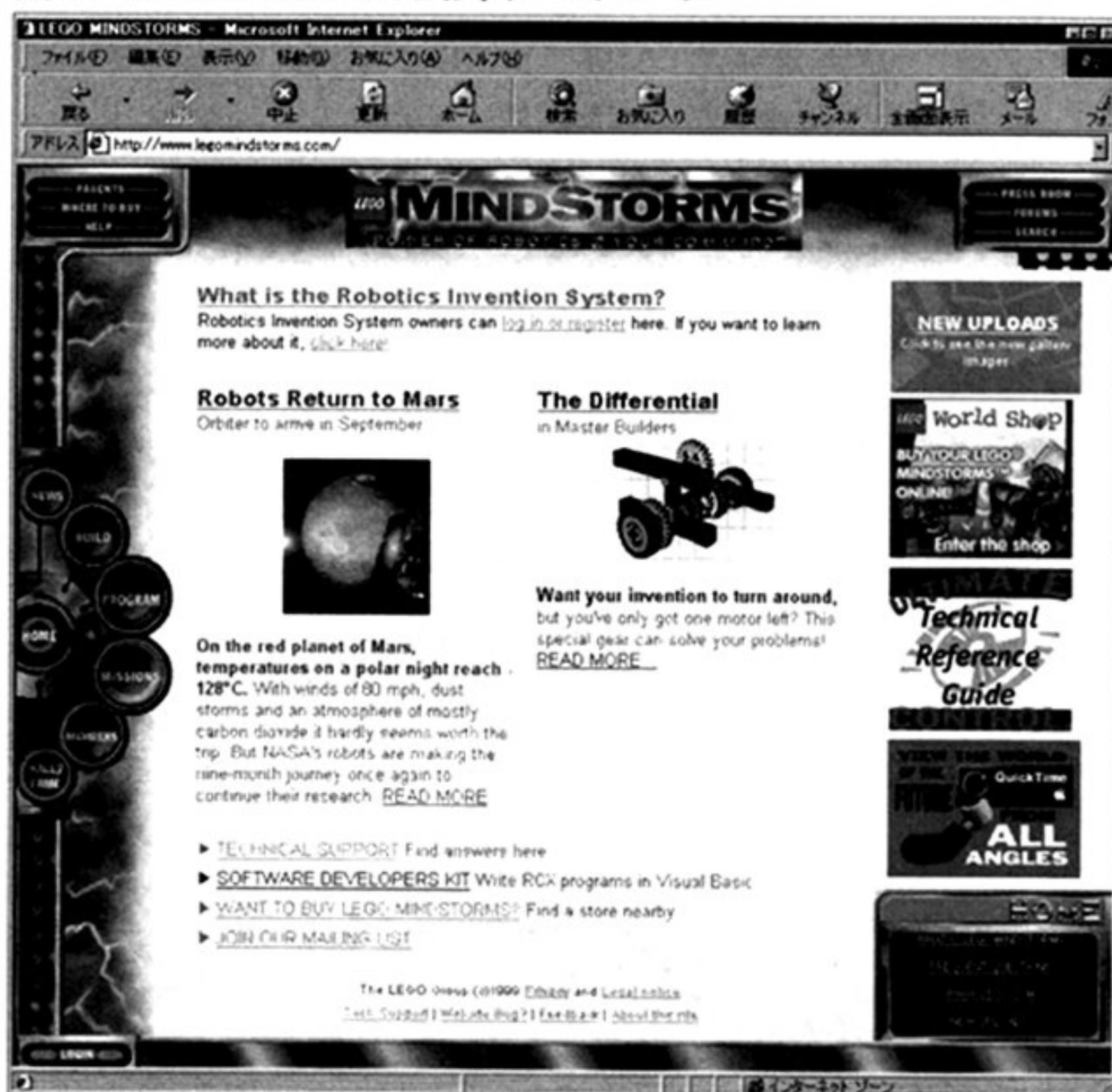



図4　MindStorms情報局

局」[5]が抜群の充実度を誇っていますので，そこをご覧ください。毎日毎日，最新情報が載りますし，氏の作品群も掲載されています。

密かなたくらみ

Papert博士の研究の到達点である贈りもの2つを前にして，私がたくらんでいることを少しお話ししましょう。

ResnickがStarLOGOを使って説こうとしていることは素晴らしい。それは創発現象を重視した新しい科学のパラダイムとなっていくに違いない。一方で，MINDSTORMSで目指しているような構築主義的な部分，つまり実物で体験しながら理解を深めていくという部分は，前面から退いてしまい，計算機の中だけの話になってしまった。両者をなんとかあわせられないものか？

というところが基本的なモチベーションです。

ということで，我が人工生命ラボラトリ[6]では，MINDSTORMSを単体で扱うのではなく，MINDSTORMSの群れを用い，しかも，人間が仕組んでコントロールするのではなく，MINDSTORMS間の相互作用から思いがけない面白い現象がマクロなレベルで創発するという現象をなんとか引き起こしたいとたくらんでいます。

集団の中でMINDSTORMSの行動を進化させたい（ブロックやタイヤをどう組むかという構造の進化は難しすぎるけど，できれば構造も進化させたい）と思うのですが，問題はいろいろあります。たとえば，

1) センサやモーターがあまり精度のよいものではないので，協調的で高レベルの行動をとらせるのは難しいかもしれない

2) MINDSTORMSを試行錯誤させて進化させるには，時間や手間ひまがかかりすぎる

3) MINDSTORMS 4っつぐらいでは無理で，もっとたくさんないと進化などしない

1)に関しては，まあ，赤外線でRCX同士が通信できるので，それほど複雑な行動は期待しないということにしましょう。2)と3)に関しては，これはシミュレーションと実際のMINDSTORMSを使った実験をうまく融合させることで解決することにしましょう。つまり，計算機の中で100台，1000台使った進化シミュレーションを行いながら，実際のMINDSTORMSをたとえば8台使った実験を行ってデータを補正しつつ，また，デモンストレーションを行うというハイブリッドなやり方[6]です。

となると，あとは，問題はないでしょうか？ あります。

4)（現時点では）学生がMINDSTORMSにのめり込んでくれない

う～む。これはなんというか，頭で考えて解決できる問題ではないので，お手上げか？ あと，これも。

5) 人工生命ラボは子供のおもちゃをせっせと買い込んでいると後ろ指を指される

でも，これは，ノープロブレムといっておきましょう。僕にとっては，研究もこの原稿書きもなんでも楽しんでやっていますし（5時間続く会議とか，出なければいいのに勉強する気もないのにボーっと座っている人が80％を超える授業とかは，さすがに楽しんでやれたら，それはそれで問題ですが），パソコンだって，研究の対象だって，まあ，基本的にみんなおもちゃのようなものなので，いまさらなんと後ろ指を指されようとも！ なのです。

参考文献）

[1] M. Resnick, "Turtles, Termites, and Traffic Jams", MIT Press, 1994.

[2] 有田隆也，"人工生命"，科学技術出版，1999（印刷中）.

[3] MITのメディアラボのStarLOGOのホームページ http://www.media.mit.edu/starlogo

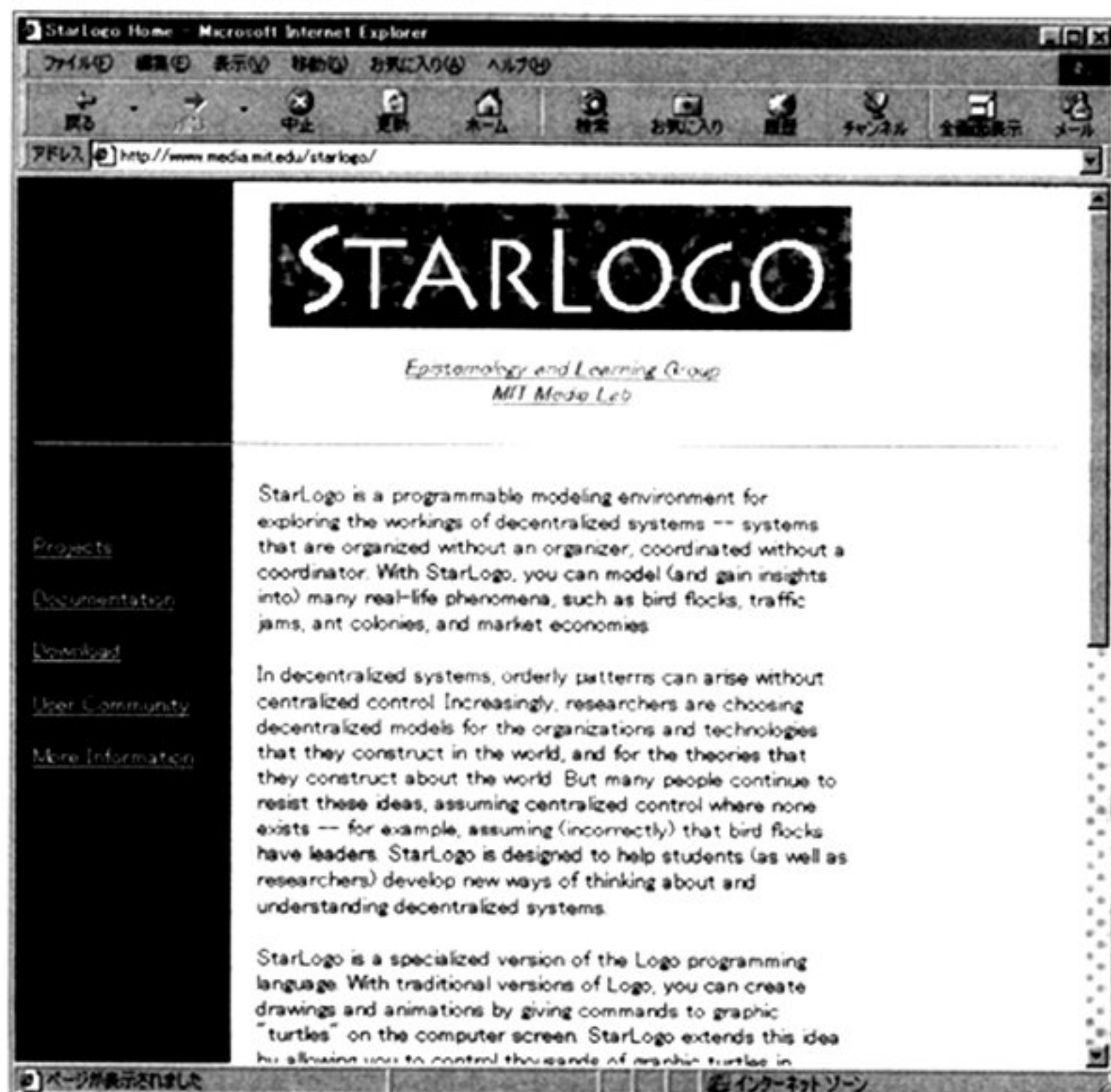
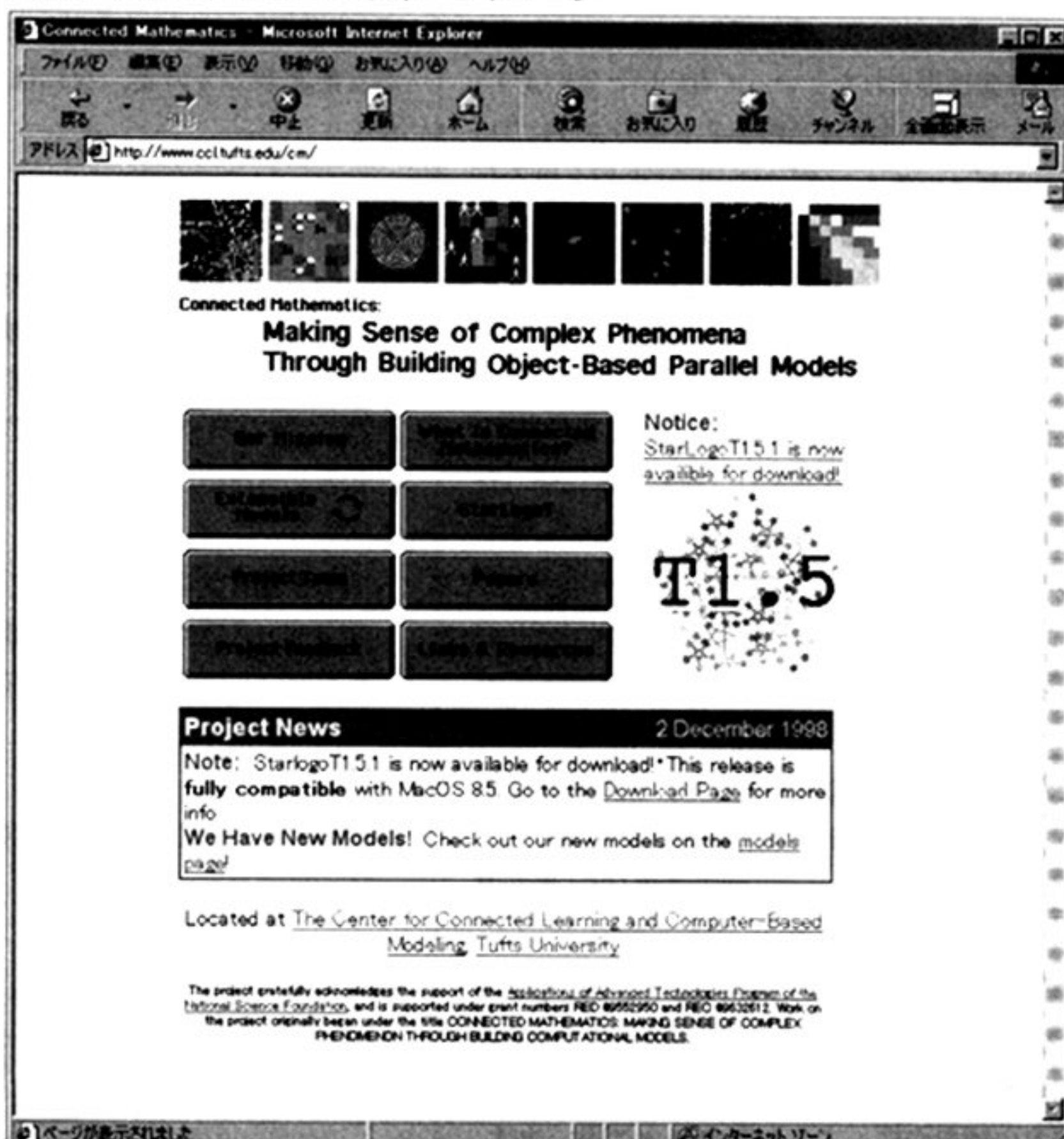
[4] Tufts大学のStarLogo Tのホームページ http://www.ccl.tufts.edu/cm/

[5] JinsSato氏運営「MindStorms情報局」のホームページ http://www.mi-rai.com/JinSato/MindStorms/index.html

[6] 人工生命ラボラトリのホームページ（未整備） http://www2.create.human.nagoya-u.ac.jp/

[7] T. Arita and C. E. Taylor, "A Simple Model for the Evolution of Communication", Evolutionary Programming V (Proc. of the Fifth Annual Conference on Evolutionary Programming), pp. 405-409, The MIT Press (1996).

図6 StarLOGOTのホームページ

図5 MITのStarLOGOホームページ

STUDIO X ON AIR

FROM READERS TO THE EDITOR

復刊のお祝いに数多くのメッセージをいただきました。どうもありがとうございます。熱い思いのこもったメッセージが多く，たいへん心強く思いました。そんなメッセージの一部を紹介しましょう。今回もコメントがあまりつけられませんでした。ごめんなさい。

illusutration：Takahashi Tetsushi

◆この読者アンケートは香港の刑務所の中で書いてたりします。昨年の春にインターネットの事件でとほほなことにインターポールから指名手配され，香港警察に逮捕されました。それで友人からOh!Xを差し入れてもらって読んだ次第です。NetXのページは見ていたので雑誌が出るんだな，とは思っていましたが，Oh!Xの復刊とは予想できませんでした。ビートルズが再結成して新曲を出してもビートルズであるように復刊したOh!XもOh!Xであったのでうれしく思いました。

さて，私事ですがいまは刑務所の中で「週刊女性」を読み，「赤毛のアン」を読み，生まれて初めてMacintosh系の雑誌を読みながらOSの勉強をしています。具体的にはMach，TRON，BSD，Linuxやらのカーネルや，X，MacOS，Windowsのウィンドウシステム，BeOS，OS/2のプログラミング環境です。本はコンピュータちんぷんかんぷんの姉から送ってもらうので思うようにはいきませんが。

数年前にはX68000の次世代機のアーキテクチャであーだこーだと盛り上がったものですが，いまとなってはAT互換機（インテルアーキテクチャなのはさておいて）で，PlayStationで，そしてDreamcastでハードウェアの理想は得られたといってもいいと思います。そこで必要になるのは復刊号のタイトルのように〝パーソナルコンピューティング〟のためのOSだと思うわけです。Human68kの32ビットDOSというのは素晴らしい環境だといまでも思います（なんでMSはWindows作る前に32ビットのDOSを作らなかったんだろう？）。

満開製作所も互換機を作る（名前はこれこそが「満開一号」なんでしょうか）そうですし，私も日本に帰ったらX68000の文化を礎としたOSを開発するつもりです。なので，春号の特集はネットワークゲームだそうですが，次の夏号か秋号ではOSを取り上げてほしいです。そのときはもしかしたらご協力できるかもしれません。

なんだか私信みたいになりましたが，次号も楽しみにしています。たぶん，日本で読めるので。

武田洋幸（25）愛知県

香港で読んでいただけるようにお送りするつもりですが，微妙な時期かも。ハード関係に比べてOS関係は開発者が少なく，なかなか手が出ない部分ですね。確かに独自OSってのも面白いとは思います。とはいえ，丸ごとやるとやはり大変なのでGUIとかマイクロカーネルとかちょっとずつ実験することになるんでしょうね。

◆熱い，熱かった。久しぶりに「Oh!X」を手にしたこと，3年前休刊になるという記事が信じられなくて途中で読めなくなったのと同じくらい熱かった。それと，Oh!Xの方向性についてですが，よくも悪くも読者が決める。私も一読者としてできる限り積極的に参加していきたい。

竹内宣博（25）愛媛県

◆久しぶりだ。こんなに真剣に雑誌を読むのも。文字を追うのが苦痛にならない。時間が過ぎるのを忘れて読みふけってしまった。毎月多くのパソコン雑誌を購読する。どれを読んでも面白いと感じることはない。パソコン雑誌は新製品かバグなどのお知らせと付録CD-ROMに入ったメーカー提供のドライバ，アプリケーションや体験版などを得るためだけのもの（極論だけど）だと思っていた。けど，違ったのですね。読み終えてパソコンでなにかをしてみよう（仕事でなく趣味で）と思わせる雑誌もあったのですね。なにか文章が支離滅裂ですけど，なにはともあれ復刊おめでとう！

伊藤浩二（31）大阪府

◆プログラミング環境の点から考えるとMacやWindowsよりもPC UNIXのほうがはるかに上でしょう。X68000からの移行先としてはいちばんだと思うんですが，ほとんど触れられていませんね。ちょっと不満。余談：「プログラミング言語C++第2版」の日本語版は名著じゃないと思う。

松井和宏（29）静岡県

UNIX系でプログラム作ってる人は多いんですが，GUIベースでやってる人がほとんどいないんですよ。X上でいろいろ環境を作っていこうとするとなかなか人がいなくって。

◆魂が燃えるぜ！　復活おめでとうございます。

畠山明良（？）京都府

◆う〜ん，おっさんばっかし。ま，しかたないか。私もおっさんになっちゃったし……。

坊農 誠（27）奈良県

◆休刊と聞いたときはとても寂しい思いでしたが，書店でOh!Xという文字を見たとき，まさか！　と目を疑ってしまいました。ぜひ月刊化してほしいです。Oh!Xの記事はほかのパソコン誌と違い，とてもためになるものばかりなので復刊はとても嬉しいです。

角川浩一（28）宮城県

◆最初聞いたときはただの噂だと思った。しかし，気になったのでとりあえず確かめる。なんと本当だった。あい変わらず，不思議な個性を感じさせる内容だったので，次も期待。必ず次も出してください。

下川将紀（27）長野県

◆Webのサイトはときどき見ていたのですが，やはり実物を目にすると驚きます。すっぱりコンピュータ業界と縁を切って，某大学の大学院生をやっていますが，私にとって，いまの「季刊」のスタイルのほうが親しみやすく感じます。とりあえず，これからも買い支えつつ趣味のプログラミングなど私も続けていきたいですね。

小井田伸雄（25）東京都
組み立てK6-2/300

◆僕はX68000ユーザーではなかったのですが，憧れは持っていました。しかし中学生だった僕には買えるような代物ではなかったんです。いろいろあって現在はゲームプログラマをやっています。技術的に日進月歩の業界人として，このように幅広くいろいろなことに関しての技術雑誌は最高だと思います。特に丹さんの記事はプログラマとして考えさせられるものが多かったです。運よく，次回作は車ゲーなので参考にさせていただこうと思います。次回のOh!Xも期待しているので頑張ってください。

徳永圭児（24）京都府

あ，丹君の記事は一部間違いがあって，そのまま参考にするとマズイかも……。

◆Oh!X復刊おめでとうございます。私はTOWNSユーザーなので，Oh!Xという本は見たことがありませんが，Oh!FM TOWNSは毎号読んでいました。Oh!Xの内容は，いまの初心者向けパソコン誌よりははるかに難しいですが，とても面白いです。あと，プラットフォームを選ばないということですので，ぜひTOWNSのコーナーを作ってください（ムリかな）。

仁科和也（23）岡山県

おお現役TOWNSユーザーですか。Oh!FM TOWNSはいつもお隣さんでした。Oh!Xでは「〜してください」はBadです。「私が〜します」がGood。新しいスタッフではMSXのコーナーを作ると張り切っているのもいます（どうなることやら）。パーソナルコンピューティングという枠から大きく外さない限り場所の提供は考慮します。他機種の人でも読めるものに仕上がっていれば問題ありませんので，なにかできそうなら投稿など検討してみてください。

◆じっくり読んでからアンケートを書きました。私がOh!Xを，というよりコンピュータ雑誌を買う最大の理由は，やはりゲームを自分で作ってみたかったからだと思います。ですから，読んでわからない記事であっても，ちょっと調べて理解できそうな内容がてんこ盛りだった一昔前の本はあとで読み返してさらに面白かったものでした。Oh!Xはそんな本だったはずですが，さすがに苦しいところではありますね。「作る」って環境がもっと身近で当たり前の世界が懐かしいやら，不思議だったやら……。 大矢英光（25）新潟県

◆復刊おめでとうございます。そしてありがとうございます。感動に震えてしまいました。これからもどんどん続けましょう！ パーソナルコンピューティングについて語れる本はこの本しかないことはみんなわかってるはずですから。知能機械概論100回おめでとうございます。やっとこさ100回記念パーティができますね。

中本昌文（23）兵庫県

◆復刊号を手にして，私がひかれていたのはX68000自体ではなくOh!Xだったのだなあと再確認しました。次号楽しみにしています。 遠藤俊夫（24）東京都

◆なにかを「作る」ことの醍醐味を，与えられることしか知らない人たちに伝えられる本になってほしい。かつてボクがOh!MZにDTMの面白さを教わったように。

中島民哉（28）埼玉県

◆最初復刊すると聞いたとき，正直いってもういいかなと思ってました（休刊するとき，もう終わりだろうなと思ってましたので）。でもやっぱり本屋で見つけたとき懐かしくて，いまこうして手にしています。次もあるみたいなので頑張ってください。

村松智行（25）静岡県

◆正直にいってしまえば，私はOh!Xが休刊したときほっとしたのだ。なぜなら，私はある人物の代わりにX68000の終焉を見届けたかったからである。パソコンというものの終焉とはなにかと問われると答えに窮する。MSXなどはいまだに使われてユーザー間の交流もあり，新機種まで出るのだから。それこそ，この世にユーザーがひとりでもいる限りパソコンの終焉などないのかもしれない。そこで私は便宜上定義した。その機種に関する定期刊行物が発行されなくなること。アスキーですらMSX関連の発行物がないのだからよい定義であると思った。ところが，Oh!Xが休刊になっても電脳倶楽部という定期刊行物があるではないか。その上，今回のOh!Xの復刊である。おいおい誰だよ不死鳥なんかマスコットにしたのは？

小川博（28）神奈川県

不死鳥はカウンタースペルがかかってますので，縁起の悪いものとされています。マスコットはツタンカーメンですね。なんとなく復活しそうなマスコットじゃないですか。

◆Oh!MZがドラゴンであったように，もっと高いレベルの記事で突き放してください。それに追いつこうとする感覚をもう一度味わいたいです。

千葉浩貴（25）宮城県

◆なんとなく買ってみましたが，結構いい感じでした（高いけど）。Oh!X68000のコーナーを読んでいるとX68000がとてもほしくなりました。X680x0互換機に期待！ 須賀さんのMoon Over the Castleの完成度には脱帽です（SMFでも入れてほしかった）。

小西秀典（18）北海道
PC-9821V12

◆Oh!Xが休刊してからいままでに数々のクリエイティブ指向な雑誌が出現しては消えていった。PC-FX GAやネットやろうぜなどのゲーム開発環境もいまではそのなりをひそめてしまっている感がある。こうした「クリエイティブ」動向が軒並み失敗に終わっているなかで起こった「Oh!X」復刊。「もの作り」大好き人間の私としては，この事件はとても心強いものだと思った。今後とも頑張ってください。

花山慎一（20）福井県

いってはなんですが，よその本と比べてもらっても困ります。パソコン誌がつぶれるのはよくあることで，それ自体はなんの関係もありません。同じジャンルの雑誌が存在するとは思っておりませんので。

◆以前ゲーム機としてX68000を使っていましたが，引っ越しの際に捨ててしまいました。もう一度プログラミングをやりたいと思ったときはハードもソフトも関連書籍もなく，手遅れだと諦めたものです。が，Oh!Xの復刊，値段も見ずにレジに持っていきました。できれば互換機が発売される前から月刊のベースでお願いします。今度こそコンピュータを扱えるようになりたいので，基本的なこともやってくれると嬉しいのですが……。とにかくOh!Xの復刊嬉しいです。頑張ってください。 綾香吉則（26）東京都

◆あ〜社会人1年目にして早くも会社が嫌になりました。忙しい退屈な日々よりも，辛くて楽しいやりたいことのある日々を送りたい……。そんなに難しいことなんでしょうか。技術系ライターなんてとってもらえるのでしょうか？ 普段はアマチュアCGA作ってます。

小林佳徳（25）東京都

ふ〜ん，と職業を見ると……大手印刷屋さん。そりゃあ辛いかも（身に覚えあり）。現在は思いっ切り好きなことやらしてもらってる身ですが，ぜんっぜん楽ではないですので，世の中うまくできてるもんです。ま，自分の人生ですから，ヤクザな道でもライター稼業でやってきたいなら連絡してください。

◆Oh!X復活おめでとうございます。休刊になる雑誌は数多くありますが，不定期とはいえ復刊を遂げた雑誌はほんのわずか。そのなかにOh!Xが加わることができるとは驚くと同時に喜びをかみしめております。アセンブラやハードの話など，あい変わらず濃い内容で嬉しいです。今後も楽しみにしておりますので2号，3号と続けていきましょう。 金山知俊（27）愛知県

PC-9821Xa7/C4改

◆昔Oh!Xを読んでいましたが，あの頃の熱そのままというか，さらに熱くなっていますね。C++かVBを始めようかな……。ぜひとも次号をお願いします。

川崎修治（31）兵庫県

◆「ゲームデザインのススメ」がよかったです。いま，ハードの進歩が速いせいなのか，ゲームの技術やスケールばかりが注目され，システムに関してはほとんど評価されることがないような気がします。なんでも詰め込んでなんでもできるのが素晴らしいゲームといった企画が多いなか，こういう話がもっと広まってほしいものです。 花井章能（28）東京都

◆あれ？「復刊」ですか？ 私はてっきり「帰ってきた」と書いてあるとばかり思っていたんですが……。まあ，なにはともあれ「お帰りなさい」そして「復刊おめでとうございます」記事のなかでも「マイナーから伝説へ」とX68000を謳われていましたが，次回はもう決定ですね。「伝説から神話へ」いやいやなんだか違いますね。「伝説からお笑いへ」こっちですね。ハハハ。実をいうとOh!Xの休刊とともに私のパソコンに対する興味も徐々に薄らいできていたのですが，Oh!Xの復刊を機に再び興味が復活するかなと思っています。でも，復刊Oh!X，1号だけってのはなしですよ。本当にね。

藤原彰人（28）沖縄県

一瞬考えたのは認めます。ただ，あの辺の位置にあんな感じで文字を置いた場合，誌名と間違えられるといろいろうるさそうなのでやめました。誌名ごと変えるという手もあったんですが，それはさすがに……。最近はJackのほうが通りがいいようですけど。

◆満開の電子ちゃんは広告ではなかったのか……。広告目次に載ってないし。 小野寺秀一（30）神奈川県

するどいですね。あれは編集ページです。絶対に欠かせない記事ですので。

◆すっげー！ Oh!Xだぁ。久しぶりー。中身もちゃんとOh!Xしてるー。うそ〜〜（31歳男のコメントかコレ）。 森山昇一（31）兵庫県

勢いのある文字をお見せできないのが残念です。宇宙人森山さんのコメントでした。

◆行きつけの書店で偶然見つけました。懐かしいロゴ。懐かしい執筆陣。懐かしい誌面からの香り。なにもか

も皆懐かしいって感じです。以前はX68000でC言語を使って自分なりのツールを作ってました。自作AT互換機を使うようになってからはVBでプログラミングしています。C使いの身にはBASICの文法，辛いです。今回C++の記事を読み，再びCの世界に戻る決心がつきました。いつもOh!Xなんです。X-BASICからマシン語に移ったときも，アセンブラからCに移ったときも，VBからC++に移るときも。きっかけはいつもOh!X。第2号楽しみに待ってます。

加藤昌宏(29)埼玉県

◆大学で1年留年のあいだにX68000XVIを手に入れ，いまゲーム会社でプログラマをやるようになっても，まだOh!Xが本棚の一角を占めています。これだけは捨てられない。いま新しく1冊加わることを嬉しく思っています。第2号，第3号とこれからも頑張ってください。

大矢 裕(25)北海道

◆復刊号読みました。結果……自分が思いっきり退化していることが判明。リトライするしかないですね。ただ，どの環境でするか，結構悩むところです。会社で，Macintosh，SUN，IRIX(Indigo^2)，家で，Win98，X68Kを使ってるものですから(まぁ，会社のは「あくまで」触ってるレベルですが)。ところで，現在，DTPの仕事をしているせいか，カラーページを見れば，「あ，版ズレ起こしてる」とか，「ブラックオーバープリントなってないやんけー」とか。まずその辺が気になってしまいます。あと，色かぶりした写真を見るとすごく気になったり。職業病ですかね。……16折ですか。

澤山繕作(25)奈良県
X68000初代，EXPERT，X68030，G6-300

「だいたい256ページ」って感じで作ってましたので，なんやかんやで16折です。こちらもまだまだリハビリ中ですので，慣らし運転状態です。突っ走れるようになるのはまだまだ先でしょう。遅れないようについてきてください。

◆この日がくるのをずっと待ってました。ここ何年かは，仕事の関係でWindowsマシンを使わなければならず少々退屈したパソコン生活を送っていましたが，Oh!Xのホームページを見つけたときには本当にびっくりしました。以前，私にとって初めての投稿プログラムが掲載されてからわずか数カ月で休刊となってしまい，当時は結構ショックを受けました。残念ながらまだ復刻記念号は入手していませんがなんとか手に入れたいと思っています。ところで，私は札幌市の近郊に住んでいますがいちばん入手しやすい書店はやっぱり紀伊国屋あたりでしょうか？　札幌って結構X68000ユーザー多いから予約が必要かな？　それではこれからも頑張って夢のあるパソコン雑誌を作ってください。

上田 剛(39)北海道
X68000XVI(ノーマル)，東芝BREZZA

Remember your dreams,
because your new dream partner is...
Comming Soon!!!!!!? >Zero Shiki

```
#include <dream.h>
#include <power.h>

main()
{
   while(MyDream != EOF) {
        printf("何時までも   ¥n");
        printf("     夢がある   ¥n");
        printf("そんな無限Loop  ¥n");
        printf("     Oh!X   ¥n");
   }
}
```

◀SHIMSOFT

by SHIMSOFT

e-mailでの投稿です。去年札幌に行ったときも全然手に入らないと聞かされました。そうですね，紀伊国屋ですかね……。多めといっても，あまり数は入ってなかった気がしますが。ムックの場合定期購読とかはありませんので，注文していただくのがいちばん確実です。

◆こんにちは，もうX68000を触らなくなって5年になります。しばらくパソコン関係から遠ざかっていたのですが，Dreamcastで初めてインターネットを知り，X68000のHPを見つけて驚きました。Oh!Xが休刊になったときは本当にショックでした。復活の文を見て久々に興奮してきました。知らないあいだにX68000も進化していたんですね。Z-MUSIC ver.3も知りませんでした。久々にX68000を動かそうかと思っています。

粟谷 茂(23)愛知県

◆復刊号は，2冊購入してしまいました。別に買い占めとかいう意味ではなく，自分の「マイコン」に対するひとつの記念として買いました。画像関係の内容については「ツールが進歩したし，画面の解像度や発色も本当に綺麗になった」と思います。でも結局のところは「描き手の才能」という部分が大きいのであるなぁということも痛感する記事でありました。

で，私はといえば，あい変わらず「マチエール」というツールをX68000で使用しております。結局は手に馴染んだツールがよいということもあります。自分の，ホームページの背景画像なども，いまだにこれで作っています(X68000用のスキャナがまだ健在で，こちらを使用しているせいもあるのです)。060 TURBOというアクセラレータがありながらマチエール用のパッチが当てられないバージョンなので，旧型機で使わなくてはならないのが残念です。Windows版が出ていないのですが，なんとかこれをフリーな形で公開してくださるとありがたい気がします。

ホームページ作成のツールはWindowsの市販品を使っていますが「こういう程度のものなら自分で作るというのがXの文化ではなかったかなぁ」と思うこの頃です。

湯澤 聡(36)埼玉県
X68030(＋060TURBO)/Compact，自作互換機

◆ダンプリストのないOh!Xってなんかやだ。

川崎睦郎(30)大阪府
IBM PC/AT互換機

打ち込みたいという要望が多ければ載せますけど？　数KBで使いものになるものを作るのは難しいかも。ちなみに，締め切りが1カ月早くなるCD-ROMよりは編集作業は楽になります。本の厚さがどうなるかは知りませんが。1ページあたり2KBとして……MFCを使ったプログラムをひとつ載せるよりはDVD-ROM2枚つけるほうが安いかも。

◆いろんなことがあった3年とちょっとでした。あの12月号は買えなかったし。探し回ってもダメでした。'95年の年末はOh!Xを探して過ごしたようなもんです。それからたびたび夢に出てきました。『Oh!X復刊！』。いつもそこで目が覚めるんです。そして3年……。復刊号を本屋で見つけたとき，本屋さんの真ん中でホントに泣きました。本屋さんで泣いたのなんて，もちろん初めてでした(当たり前か)。NetXも知らず，オフラインの愛機(ACE-HD)を前に無為に時間を浪費する毎日でした。「インターネット？　それがどーした……」。思えば，ひねくれた3年間だったように思います。どんどんと周りから取り残されていった自分でした。が，いまや人様にパソコンを教えたりしている身分。しかも，Oh!X復刊。こうなったら他人にインターネット接続を教えている場合ではありません。少しずつですが，リハビリを始めました。CG描くべし，でマウスをこねくりまわし，MIDIで作曲も始めました。久々に，本当に久々に少しずつ「創る」夢を見始めています。Xユーザーとして他力本願では駄目なんだがなぁと思いつつも，やっぱりOh!Xは偉大だ，と痛感する毎日です。いつか，リハビリの結果を発表できる日を『夢』見ています。

PS.我が家のACE-HDはまだまだ現役だったりします(笑)。

Woody_Cat(31)大阪府
X1turbo，X68000ACE-HD，HB-F1XD(MSX2)，FMV-DESKPOWER T4267

◆昔はOh!X＝X68000ということで買ったことはなかったのですが，今回「復刊」という文字を目にして手に取ってみたところ，内容が面白そうだったので買ってしまいました。面白いですね。私が求めていた雑誌はこれです。2号目にも期待しています。

松本智行(23)東京都

◆最近，近所の粗大ゴミ置き場でX68000PROを拾った。DOS/Vユーザーの私にとっては，X68000はパソコンの知識をつけようと読んだ本に「過去に威勢のよかったゲーマー用パソコン(パーソナルワークステーション)」と書かれていて，単なるゲーム機(PC-FX)としか思わなかった。しかし，あるホームページで，「X68000復活」とか，「某製作所が互換機製作」とあったので，しばらく家に置いておくことにした。でも，本体しかない。その本体は傷だらけ，そして，インターネットできないの粗悪三拍子がそろって，なにに使われるのか知りもしないまま，倉庫の中で思い出になろうとしている。「これでインターネットができたら，部品買う気にもなれるんだけどなー」

井上 昂(14)千葉県

所詮は過去のものです。君たちの世代は君たちの手で伝説を作っていくべきでしょう。インターネットは「つながる」ものではなくて「つなぐ」ものであるこ

ととか、つながってるマシンなんて1台あれば十分なこととか、世の中、見方を変えると様相は全然違ってきます。最近の子は恵まれすぎてるんですかねぇ。

◆現時点ですべてを読みこなすことはできていないが，昔からそうだったので気にしない。あとから役に立ったりするし。人手不足ということで力になりたい気もするが，いまは自分のなかの整理がついていなくてすべて中途半端。なんとか現状を打開しようと思っているのだが……。プログラムもこれといって完成したものはないし，文章も書けない気がするし(小学校のときは毎回残された)。ウリになるものを身につけて出直そう。
清水弘和(22)東京都

小学校の作文が得意だった人なんているんですかね？　全然関係はないです。なににしても，もっと自分に自信を持てるようになるのがいちばんですか。能力ではなくて単に気の持ちようということですけど。なにかやろうとする意欲がもっとも大事です。その辺を特にパワーアップしてきてください。

◆Oh!X復刊本当におめでとうございます。書店でOh!Xを見つけたときは本当に感動してしまいました。内容のほうはVery Goodだと思います。旧Oh!Xに比べ，機種に依存しない分，旧Oh!XよりもOh!Xらしい内容になっているのではないでしょうか。この路線で2号，3号……と続けてもらいたいと思います。厳しい環境だとは思いますが頑張ってください。
伊藤 仁(31)愛媛県

◆初めてOh!Xを購読しました。パソコンに関する知識があまりないのでとても勉強になりました。まるでパソコン(特にX68000)のための教科書のようで，買った価値を感じました。これからも購読させていただきたいのでOh!Xの存続を願っています。
井関政孝(23)大阪府

◆ついていけない記事でもいいんです。ついていきたくない記事でなければ。　村上 学(24)埼玉県

◆よい本です。書店で見かけたとき「いまどきOh!Xなんか出てるよ，懐かしいなあ」と昔のよしみで買ってはみたものの，正直に白状しますと全然読まなずに年明けまで本棚に並べたまま手にも取っていませんでした(実際，最近のパソコン雑誌は買ってもちょっとしか読まないものが多いのです)。ある日，作業の合間に読み飛ばすつもりで開いたら，そのまま隅から隅まで一気に読んでしまい，1月下旬のいま愛読者ハガキを書いています。かつてあの薄っぺらい月刊誌に七色の可能性を夢見ていた自分はまだ残っているのでしょうか。続刊を期待しています。　岩熊展史(28)福岡県
自作AT Celeron/266

◆復刊の話を某誌で知ったときには驚きました。Oh!Xが休刊してから，それまでOh!Xを買いに本屋に行く夢を数十回見てきましたが，正夢になる日がくるとは思ってもみませんでした。本の中身についてですが，X68000の記事が少ないのはしかたないとして，徐々に役立ちそうな記事が多いところがOh!Xといったところでしょうか。GTの批評記事があったかと思えば，そのGTのプログラマ本人が別のページに書いていたりと面白かったです。CD-RWのドライバが書けるような連載がほしいなと思ったのは僕がX68000ユーザーだからでしょうか。
増田和通(25)静岡県

◆まだまだX68000を使っている方が全国にいて嬉しいです。私も長く使ってやらねば。同人サークルT&H PROJECTSさんのゲームは私も全部(X68000用)買っております。次回作も待っています。Pメーカー2，プログラムは書けませんが，シナリオのようなものは書いてみたいです。　井田幸一(38)長崎県

◆書店でOh!Xを見つけて目を疑ったね。「まさか本気だったのか」内容も本気を感じさせる仕上がりで満足。内容を見ると自分も参加したくなるが，レベルの高さにはタジタジ。でも，参加したくなるというスタンスはあい変わらずで素晴らしいと思う。私も頑張ろう。
尾ノ上卓朗(30)鹿児島県

◆やはり定価が高すぎると思います。これではOh!Xを読んでいた人しか買わないかもしれません。カラーページを減らしてでも価格を千円台くらいに下げてほしいです。内容は私には難しいものが多かったのですが，楽しめました。ぜひ次を早くお願いします。
中島竜大(26)福岡県

高いですよね。やっぱり。それは重々わかっているのですが……。いろいろやっても安くなりそうになかったので，今度のOh!Xは思い切り豪華に作ってあります。逆効果という話もありますが，半分趣味で作ってますので。これでもうちょっと制作に時間をかけられれば最高なんですが……。

◆やっぱり2500円は高いです。せめて1980円にして自社の広告をたくさん入れてマイクロソフトの広告を入れれば価格は下げることができると思います。Oh!XだけでなくてMSXとかも内容に入れてほしいです。いっそ古いパソコンを全部記事にしてみたらどうでしょうか？　防人瞬華(25)福岡県

広告が取ってこれるくらいなら苦労はないんですがねぇ。ウチの広告部隊は働いてませんからねえ(とか書いてるとあとで怒られるんだが)。広告は入らないものとして計算してますので，値段はしかたないですね。それから，別に古いものを選んで扱ってるつもりはないのですけど……。

◆噂は知ってましたが，いま住んでいる山の中ではOh!Xは手に入りませんでした。久しぶりに東京に出たおり，日本橋丸善にて買うことができました。なかなか昔のようにパソコンに時間を割けない今日この頃。なにかまたやってみたいものです(ネットワーク作りに遊んでいますけど……)。　森山知己(40)岡山県

今度はSWORD森山さんですね。お久しぶりです。なかなか手に入りませんか。うーむ。しばらく本業に専念されるとのことでしたが，一段落ついたらまたパソコンのほうにも手を出してみてください。それまでにいろいろ基礎環境が作れればいいのですけど。

◆なんだとー。いまの世の中ブラックや縦型のCPUが珍しくなくなったと思いきや今度は5色のパソコンだとー。その昔，レッド，シルバー，ホワイトの3色でデビューしたX1を思い出し，ムックを前に，もはや気分は"荒城の月"。したがこのムック，我らが厳窟王(U)氏の英名をば永遠に称えるべくかの人の名をば記し，もって我が子孫への手習本といたさん。……よもや毎月発行とはあいなるまいて。　奥時雄(63)大阪府

いや，別に厳窟王とかいうわけではないのですが……。えっと，シャープの奥部長(元)ですね。商品企画の鳥居氏と両輪でX部隊を牽引していらした方です。X68000芸術祭などで前面に出ていらしたのでご存じの方もいるでしょう。現在は揃って定年退職されてちょっと寂しい限り。かなり無茶をして出している本なので毎月発行は絶対に無理です。

◆Oh!X復活おめでとうございます。休刊だったとき読む雑誌がなくって(特にX68000系)。X68000系はこれしかないと思ってましたので嬉しかったです。ありがとうございます。　岩瀬裕子(25)東京都

◆休刊になってさまざまな他誌を読んだが，肌にあわなかった。Oh!Xの休刊というのは，他誌の実質廃刊とは違う気だけはした。今年(昨年です)になっても満開はつぶれない。ソフトもハードもまだ作ってる人がいる。7月からインターネットを使えるようになると，EX68，互換機計画，そしてOh!Xの復活を知る。みんなどこかおかしいぞ。　尾形哲夫(39)北海道

◆私は業界人ではないので丹氏の記事を自分のスイング(ゴルフ)を思い浮かべて読んでいます。筋肉からシミュレートするゴルフゲーム……いいねぇ。いま30代に入っちゃった人はほとんど読んでいないのでしょうね。皆，居場所がある。雑誌にアンケートハガキを出すこと自体が久しぶりです。なんだかとても嬉しい。
家田貴之(30)大阪府

いえいえ，30代の方多いですよ。それにしても筋肉から動かすゴルフとはなかなかよい発想ですね。

◆置いてあるかと不安を感じつつ本屋に行くと，どーんと平積みになっていました。数日後にはほとんどがなくなっており，なんだか嬉しくなりました。いまのパソコンは職場で必要だからとか，なにかしかたなくという，まるで学校で勉強するのは受験のためというのと同じ感覚に思われます。そうではなくて，パソコン自体に興味を持ったからという場合，そのユーザーの集う場となる雑誌は見かけません。そうしたユーザーのためにも本格的な復活を目指してください。少しでも協力していきたいと思います。
石川貴康(27)富山県

◆Oh!Xを初めて購入しました。なんだか熱い雑誌ですね。頑張ってください。X68000シリーズは持ってないのですが，満開製作所から売り出されたら買おうかなと思いました。　高橋渉(23)栃木県

編集室から from E・D・I・T・O・R

SHIFT BREAK

▶Oh!X復刊2号おめでとうございます。今回、初めてCGと記事を執筆させていただくことになりました。今思えば、私がX68000を使っていた頃は、ただの一読者だったのですが、このように、Oh!Xで記事を書かせていただけるのは、本当に夢のようです。実は私は、Oh!Xに限らずライター(?)としてのお仕事は初めてなのです。最初は、私の制作したCGだけでも使っていただけないかと、思っていたのですが、(U)さんから記事を書かないか、というお話をいただいて、今回お引き受けしました。CGだけでなく、記事まで書かせていただいて、とても光栄に思っています。原稿のお話をいただいたのは、昨年の11月頃で、原稿の締め切り日は、年内と聞いて大あわてで記事を書きました。そのため、内容のほうは少し混乱してしまっています。ほかのライターの方々のレベルの高い記事の中で浮いてしまわないか心配です。ところで、今回、LightWave 5.0に関して記事を書かせていただいた立場上このようなことを書くのはまずいかもしれませんが、新たな作品制作に向けて、ほかのソフトを探しています。実は、もうすでに目をつけているソフトがいくつかあるのですが、特に、人物制作用にモデリング機能が高いとなれば、そちらに移行しようかなと思っています。長くなりましたが、これからも、Oh!Xの活躍を期待しています。私も、力になれることがあれば、微力ながら応援させていただきます。それでは、これからもよろしくお願いします。 (田中)

▶今回はお休みのつもりが、土壇場で書かせてもらうことになりました。原稿を書き上げた日付は怖くていえません。パソコンをめぐる物事を、少し変わった切り口で語ってみたいと思っているのですが、今回は少し滑ったかなぁ。パソコン雑誌の片隅に、少しはこういう記事があってもいいんじゃないかと思っていただければ幸いです。 (OGA)

▶通信をするゲームは昔からあるが、いまだに必要不可欠な要素としてそれを主軸に据えたものは少ない。通信に必要なものが初めから備わっているドリームキャストでも、なんでもいいから実際に一度以上通信したという人の数は、買った人の約半分にすぎない。しかし、ゲームというメディアの双方向性をより生かすために、通信は、本来中核をなすべき技術ではないだろうか? そういった意味では、ここのところ意気消沈気味な業務用ゲームに、通信ゲームの未来を見せてほしいと思う。 (如月)

▶犬を飼っています。おとなしいピレネーのおっさん犬なのですが、家に誰もいなくなってしまうと寂しそうに鳴くんです。かと思いきや、竹竿売りや広報車の音を聞くとなぜか遠吠えを始めます。おっさん犬も大変だなーと思いつつ仕事をしています。僕の犬ではないんですが。 (由水)

▶今回のFlipper Pinball Stories、MSピン、海外コインオペゲームにおける通信を今回は担当しました。当初はフリッパーおよびその近辺について紹介しようと思い執筆を開始しましたが、完成した原稿の量が膨大すぎたので今回はまずプランジャーについて記してみました。もっともっと資料的価値のあるコーナーにしたいと思い、現在さらに資料を集めているところです。回を重ねるたびに密度が高く、多くの人にわかりやすいページになるように努力します。この場を借りて私の経営するマインドウェアのHPを紹介します。ピンボールについてのさまざまな情報を用意しています。ぜひご覧ください。

http://www.mindware-corp.com

e-mail: albert@mindware-corp.com (市)

▶前号からお世話になってますが、まさか,また4コマをOh!Xに、しかもカラーで掲載してもらえるとは、人間長生きしてみるもんです。本誌の記事のように高度なことは書けませんが、華を添える気持ちでOh!誌キャラ総出演で楽しく描いて参ります。よろしく。 (直)

▶非常にタイトなスケジュールのなかで原稿を書くことになった。前回、丹さんと横内がうねうね書いていて、なにもやらないくせに、打ち上げだけは行ったので、ちょっと気まずさが残った。よし、さすがに次は手伝おう、と……。しかし、いままでのように、先にやっておけばよかったという次元ではなく、いまの私の場合、いつ、この修羅場を持ってくるかでしかない。つまり、いつも忙しいらしい。忙しいことはいいことらしいのだが、これで倒れたら、元も子もないような気がするんだな。で、私が死にそうになって書いているときに、丹さんが、次は手伝おうということをいっていた。きっと前回の私と同じ気持ちに違いない。ただ、丹さんは前回、ひーひーと次はやらないとかいってたよ～な気がするのだが、喉元をすぎれば熱くないのだろう。私も、次はやらないもん。 (瀧)

▶ボイスラッガーにハマっている。わけのわからないノリ、素人くさいアフレコ、しょぼい絵、どれをとっても「うんうん、よくできた自主制作映画だね。どこのサークル?」って感じだ。三脚もないみたいで手ぶればりばりだし、集音マイクすらなさそう。きわめつけは、夜中にそこらの道端で突貫撮影しましたって感じの現場。すごいよなぁ、よくこんなのが企画通るよなぁ。さすがテレ東。なんでこれで水木一郎とか神谷明とか中川亜紀子とか関智一とか使えるかなぁ、って、この人たちってひょっとして名もないヒラ役者よりもギャラ安かったりするのかな? ま、これが売りだし。惜しむらくは1クールで終わっちゃうことか。「いつも心はシュバババババーン」って夜中に絶叫したら、近所迷惑だよな、やっぱ。 (I.K)

▶かまたゆたかです。DoGAの近況を報告します。

L3をお待ちの皆様、すいません、全然進んでいません。とりあえず先にL2をシェアウェア化します。X68時代はカンパでやっていけましたが、Windowsユーザーにはカンパという文化は理解されなかったようです。残念です。でもL1はフリーにしますので、とりあえずCGを始めたい方への門戸を開くという当初の目的は損なわれません。L1、L2の英語版が完成し、海外に配布を開始しました。でもまだまだ知名度が低く、反応はあまり多くありません。4月7日からアマチュアのCG作家をメジャーデビューさせる「D's Garage21」という番組がテレビ朝日で始まります。当方も作家の権利を守る立場で参加しています。見てね。 (かまた)

▶おお、今回はちゃんと編集後記が書けるぞ。前回はちょうど後記の締め切り時期にはばっくれてタイランドに行っていたので書けなかったんですが、今回はちゃーんと本文の締め切り前にベトナムに行ってきちゃったもんねー。どうぞ次回は本文の締め切り頃にどっか行ってて原稿が書けなかったりしませんよーに(苦笑)。どうでもいいけど、海外に行くたびに龍山だ、パンティップだ、とパソコンショップの集まる界隈に行っていたことに気づくと結構悲しいもんがあります。ついでにアジアのパソコンショップが集まるあたりには、たいていアニメショップもある、ということにも気づきました。いやぁ、まさかタイにきて等身大綾波ポスターに出合うとはねぇ。来年行ったらマルチがいるかしら。ところで、つい先日、OCNエコノミーを引いた友達の会社に、私のLinuxマシンを強制的に設置させることに成功しました。サンプルCGIやらは次から自分サーバに載せておくことにしますんで、実際に皆様に使って、機能をお試しいただけることでしょー。 (で)

▶会社を辞めて半年たった。編集後記でもこの間のことを書こうとも思ったのだが、やめておくことにしよう。というわけで、話は変わって花粉症。実は私は25年以上のキャリアを誇る筋金入りの花粉症である。おかげで、薬の箱の裏にある、「d-マレインサンクロルフェニラミン」とか「塩酸フェニルプロパノールアミン」などという、舌を噛みそうな変な薬品ともずいぶんと親しくさせていただいてきたわけだが、とうとうピーク時にはこれらの入った抗ヒスタミン剤もあまり効かな

某月某日
タブレットに
おもいっきり
ビールをぶちまける…

また

次回

すべての力を いまここに

復刊号では多くの反響をいただきました。また，スタッフ募集や投稿募集などでもさっそくいくつかの応募をいただき，心強く思っています。今回掲載されている記事のなかにもそういった投稿によるものが含まれています。

以前にも書きましたが，Oh!Xは読者の力が作っていく雑誌です。こちらから用意できるものはきわめて限られたものでしかありません。Oh!Xスタッフと読者のコラボレーションでこそ，新しい潮流は作られていくものでしょう。それは昔も今も変わらないものだと思います。読者の皆さんの積極的な参加をお待ちしています。

また，昔スタッフだった方からも連絡をいただいたり，ハガキのなかに見覚えのある名前を見つけたりと，いろんな方々とまた出会うことができました。皆さんのなかにも今回また懐かしい筆者の名前を見つけた人もいることでしょう。目次には懐かしい名前と新しい名前が並んでいます。そのどちらもが新しいOh!Xの原動力となります。

現在は少しずつ以前の力を取り戻しつつ，さらに新しい力を取り込んでいるところです。いろいろ十分ではない点も多いと思いますが，そのときにできることはすべてやるという方針でできるだけのことはしていくつもりです。

新しいOh!Xの姿はまだまだ確立しているわけではありません。おかげさまで，今後まだ何冊かのOh!Xを発行することが決まりました。今年度は3冊以上の予定です。

ただ，こちら側のキャパシティ不足で，それらにきちんと対応できていないのが非常に悔やまれるところです。現状では編集側の力不足が目立つ結果となっております。たとえば，“LIVE in '99”……投稿作品はあったのですが，CD-ROM収録などの許可取りの手配がつかず見送りとなりました。なかなか手が回りませんで申しわけありません。

スタッフ募集は常時受け付けていますが，実質的に考えると，スタッフはこちらに直接こられる方か，e-mailで連絡がつく人でないと難しい感じです。記事投稿などもあわせて募集していますので，「とにかくなにかやりたい」という方はご相談ください。まだまだ多くのジャンルで多くの人材を求めています。内容ももっともっと多岐にわたったものにしていきたいと思っています。

くなってしまった。ピーク時でなければそれなりに効果はあるのだが，眠くなるということなので，車の運転前には使いにくい。ということで，見つけたのが漢方の小青竜湯。これともう1種類よくわからないけど，やっぱり漢方薬。ピーク時にはやはりだめだが，通常時なら抗ヒスタミン剤のように鼻の中も喉もカラカラに干上がるようなこともなくてなかなか快調である。西洋薬が「抑える」のに対して漢方は「整える」という発想だからだろうか。いまのパソコンはノイマン型に見られる還元主義，ソフト/ハードという二元論，CPUという一神教的な面など，きわめて西洋的な発想の下に作られるのだが，もし東洋的な発想のコンピュータというのが考えられるとしたらどんなものになるのだろうか。そんなことを考えていたら梅が散って，もう桜の季節になっていた。　(Kuw)

▶久しぶりの編集後記。ちょっと感動。さて，最近世間ではMercedの失敗を予測する報道が多くなってきたように思う。Pentiumのラインアップにも革新的なアイデアはないし，インテルの独走にもかげりが見えてきたか。MIPS応援派の私としては非常に嬉しい限りだ。最近のニュースで驚いたのは，SunがSPARKとpicoJavaのHDLを無償で公開するというもの。picoJavaはともかく，誰もがSPARKの改良版を開発するチャンスができれば，その筋の人が恐るべき性能のMPUを作ってしまう可能性もなきにしもあらず。でも，私としてはPS2のDSP機能の命令セットが知りたいと思う今日この頃。

(幸恵ちゃん，見てる？　のAN)

▶劇薬飲料が新時代を迎えました。大塚から出ている『とんがらC』とゆーのが要チェックです。ラベルには“清涼飲料水”と書いてありますが，飲んでも清涼になりません。以上御報告まで。　(M)

▶「お仕事はさみしがりやさんだから，忙しい人のところに集まるのよ」というわけでもないのだろうが，ちょっと年末年始は異常だった。「これ以上，凄い進行はあるまい」と思っていると，それがいとも簡単に更新されてく。無理がたたって，当初4本予定していた特別企画のうち，2本を落とさざるをえなくなった。さらにやむなく発売日も延期。無念。　(U)

▶丹です。「シミュレータ指向でゲームを作る」の一部で間違いを書いてしまったのでここに訂正し，お詫びすると同時に指摘してくださった如月緑氏に感謝したいと思います。

213ページの1段目の5行めから14行めまでと，図9の右半分を次のように訂正してください。

<修正後>

図9のように物体が軸に固定されていないと，その物体には力Fにより加速度が生じる。これは力の与え方によらず(物体の重心(質量の中心)から逸れるように力を与えても)そうなる。回転運動については，トルクの支点はその物体の重心になる。支点-力点間の線と垂直な成分(F2)がトルクになるのは軸固定の場合と同様である。

以上のように，物体に与えられる外力からは回転運動を起こすトルクと直線運動を起こす力とを取り出してシミュレーションを行わなければならない。

物体に加えた力は，その一部がトルクに使われるのと同時に「すべてが」並進運動に使われます。が，ここで私は勘違いをして，並進運動には「トルクに使われる以外の成分が」使われるという手の込んだ説明をしてしまっておりました。こいつは剛体力学のかなり初歩的な部分で，そこをミスしたということをシミュレーション指向のプログラマとして恥じております。

幸いにして，自分が手掛けた力学シミュレーション(要するにグランツーリスモのことですが)では，私はどういうわけかこのミスを犯していませんでした。たぶん問題が具体的だったので直観的に正解にたどり着けたのでしょう。

グランツーリスモの話が出たついでにもうひとつお詫びしておきますと，旧Oh!X誌の連載「ハードコア3Dエクスタシー」の自動車力学シミュレーションへの道はグランツーリスモにはほぼ通じておりません。Oh!Xが休刊してから理論をほとんどすべて再構築しているのです。共通項は設計方針や理念くらいのものでしょう。なので，くれぐれもあの記事を参考になさらぬようお願いします。当時は嘘を書いていたつもりはないのですが，あれを延長したところで目標にはたどり着けないことでしょう。

力点-支点(重心)を結ぶ線
物体の運動を変化させる成分
F
力点
F_2
力点-支点(重心)を結ぶ線に垂直な成分
重心(トルクの支点)
トルクとして有効な成分

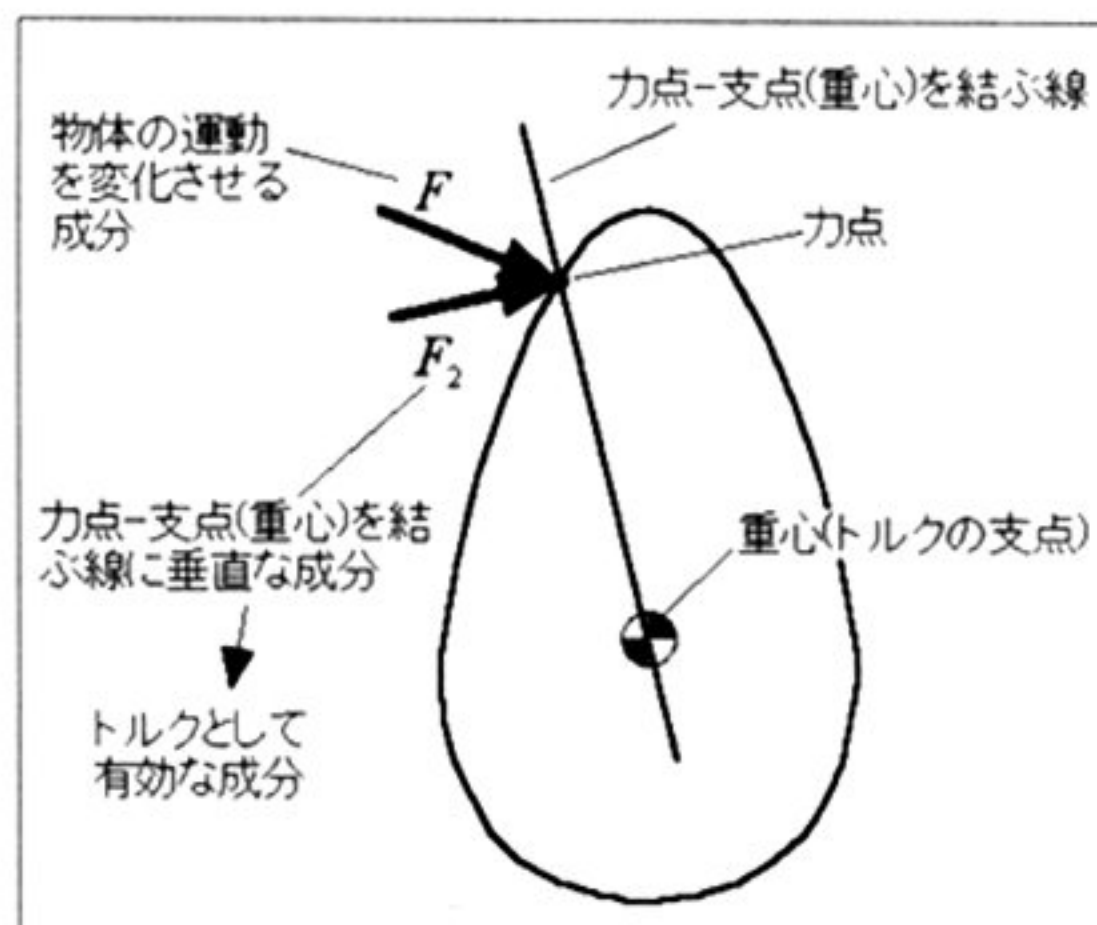



図9の訂正

全体の重心の位置ベクトルxは，各部品の重心の位置ベクトルをx_iとすると

$$x = (\Sigma m_i x_i) / (\Sigma m_i)$$

で計算できる。たとえば左のような場合，

$$x = (m_1 x_1 + m_2 x_2) / (m_1 + m_2)$$

となる。

図15の訂正その1

いずれの場合でも

$$I = m_1 r_1^2 + I_1 + m_2 r_2^2 + I_2$$

で計算できる。両者の違いはr_iの値のみ。

図15の訂正その2

microOdyssey

本誌の校了を迎えた4月2日。祝一平氏が逝去された。氏は満開製作所の経営を引退して療養中だったのだが，久々に執筆された本誌の原稿が遺稿ということになった。

氏はOh!MZ創刊時からのスタッフで「有田隆也氏に引きずり込まれた」というOh!MZスタッフではありがちな契機で誌面に参加，以降，満開製作所を設立し，そちらに専念されるまで本誌の主力スタッフとして技術面，精神面での柱として活躍していただいた。

私がOh!MZに配属されたときにはすでに筆頭ライターとして活躍しており，試験に出るX1はいちばんの人気連載だった。

それ以前の経歴にはあまり詳しくない。CP/MでMACRO80とMINCEを愛用し，アセンブラはハドソンの「Z80の神様」(Hu BASICの開発者としても知られる)に「タコ部屋」で一気に叩き込まれたという。マシン語なんて数日でマスターできる，と後日語っていたので，かなりハードな修行だったのだろうと推測される。

初期の混沌としたOh!MZ時代から「シャープに爆弾を仕掛ける会」などいろんな活動をしたのちに渡米し，パソコン用ソフトの開発に従事。いくつか手がけたらしいのだが，すべてのタイトルは私も把握していない。代表作は麻雀ゲームらしい。

「マシン語ですか？」

「Forthで書きました」

珍しい麻雀ゲームである。

私が編集部に入った頃にもめていたのがFM音源に関する解説だった。当時のX1用FM音源カードは，ボードとスピーカ，そしてユーティリティのセットで販売されていた。ユーティリティは音色をエディットし，シーケンサに掛け，リズムパターンを編集するというツボを押さえた3つの機能を持っていたものの，それ自体で完結しており，作ったデータをほかで使うことができない。X68000用MIDIボードのソフト抜きでポート番号とかの資料だけ入っているというのと同じくらい無茶な商品だった。どうもシャープさん，音楽関係は弱かった。

「BASICから使えないと意味がない」

技術的な話を書いても使えないのではしかたがない。やるなら最低でもBASICから使えるようにしないと話にならない。しかし氏はシステムへパッチを当てるなどの行為を禁じ手にしていた。BASIC内部は「聖域」で「そういうことをしてはいけないんです」という論理でメーカーがやるべき分野には踏み込まないという分をわきまえていた。

ところがシャープにはまったく動きがなく，FM音源はゲームでしか使われない。

ある日突然，氏は，

「やりましょう」

といった。

「連載でやるんですか？」

「そうです」

肚を決めてからの仕事は速く，確実だった。

X1用BASICのフリーエリアは20KB程度で，他機種より豪勢な8音を出すOPMを使ったカードは重かった。ゲームなどでの使用にBASICからの呼び出しはないと，ばっさり音楽専用に割り切った構成が取られ，G-RAMがバッファに使用された。書き込みを分割することで，非常に長大な作品も無理なく作ることができた。バッファに叩き込んでから演奏を開始するという形式はX-BASICやOPMDRV，さらにはZ-MUSICにも採用されている。

ついに発揮されたX1用FM音源ボードの性能は素晴らしく，音に飢えていた人々を魅了した。Z-MUSICに至るXのMML文化への一歩だ。

満開製作所設立も突然だった。

「Oh!X以外にそういうものが必要だ」

という洞察で私財をなげうってディスクマガジンを始めた。遠い昔に，

「広告も出さずにモノが売れるわけがないではないか」

と記事中で書いていたことがあったが，厳しい状況でもOh!Xへの広告は欠かさなかった。

また，SX-WINDOW ver.1の登場時はとにかく文字表示が遅かった。いろいろやってるので，ある程度しかたないのではないかというと，

「こんなに遅くなるわけはないんです」

と受け付けない。

シャープの人の前でアルゴリズムを聞いて，

「こうするだけでループが1/4で済むじゃないですか！」

とおもむろにコードを示していた姿を思い出す。

なお，「試験に出るX1」の著者略歴は，バイト順をひっくり返したうえで逆アセンブルすること。もちろんZ80コードだ。

「この記号はなんですか？」

「え！ 旦那，Dcケーシー知りません？」

心より氏のご冥福をお祈りする。 (U)

次回予告

Oh!X 1999 初夏号 6月中旬発売予定

特集『2次元系グラフィック』

Hardware Geeks

Immersive Spaceへの挑戦
Oh!X流デジカメ活用術(マネしちゃだめよ)
その他いろいろ

付録CD-ROMたぶん1枚
予価：未定(若干低め)

Oh!X 1999 秋号 1999年秋発売予定

内容……未定

(予定は都合により変更されることがあります)

Oh!X 1999 春号
DOS/V magazine編集部　Oh!X制作実行委員会 編

■1999年5月14日発行　定価2800円(2667円+税)
■発行人　岡崎眞
■編集人　高橋憲一郎
■発行所　ソフトバンクパブリッシング株式会社
〒103-8501　東京都中央区日本橋箱崎町24-1
編集部 ☎03-5642-8189 FAX 03-5642-3429
販売局 ☎03-5642-8100
■デザイン　クニメディア(株)
■印刷　クニメディア(株)

乱丁，落丁，CD-ROMの破損はお取り替えいたします。小社販売局までご連絡ください。

雑誌65814-57

Oh!X 1999 春号

愛読者カード

郵便はがき

料金受取人払

日本橋局承認

3181

差出有効期間
平成12年5月
14日まで

103-8730

346

（受取人）
日本橋郵便局
私書箱358号

ソフトバンクパブリッシング（株）
DOS/V magazine 編集部 内
Oh!X 1999 春号 係行

フリガナ お名前	年齢 歳	性別 男・女
ご住所 〒		
e-mailアドレス：　TEL URL：		
勤務先名（学校名）　使用機種/OS		パソコン使用歴　年
購読パソコン誌		プレゼント希望番号

1999春号通巻169号

満開の電子ちゃん

作・え 岡村祭

これ僕のアドレス
何かあったらメール下さい
わかりました

あれーっ丸がうまく描けないよぉ
まいちゃんやそんな時は円ツールを使うといいんじゃよ
……

iMac欲しいんだよねストロベリーの
えーライムの方が可愛くない？
時代は来た!!

テレビのゴールデンタイムにもパソコンのCMがガンガン流れてる
この勢いに乗じて満開製作所の製品を売りまくるんだ!!
パチパチ

そんな時には「激光電脳倶楽部」と「月刊電脳倶楽部の保存版」
どちらもX68専用のディスクマガジンでゲームやツールや音楽が満載のCD-ROMよ!!

その上皿まわしにも工合がいいの
あっ寝グセついてる
まあ 鏡としても便利なのね
ひとの話きいてた?!

私のおじさんが会社のパソコン研修で先生に
ではマウスでカーソルを動かしてみましょう
って言われて
カーソル動け!!
カ
世代だ世代!!

たとえばここにいるみんなにはわかるはずだ
「このCD-ROMを使用するためにはSCSI2規格のCD-ROMドライブと2MBのメインメモリが必要です」
わかる
わかるぞ
おー

さっきの「エックスロクハチ」ってとこだけわかんなかった
俺も
私も
いつもより余計に廻しております!!

岡村祭センセはそのうちｉＭＡＣ導入のご予定との事。マジでそろそろパソコン買ってね！打ち合わせに困るのだ！(担)

月刊電脳倶楽部の保存版CDはこの春にリリース予定です！

みかぜ（東京都）

みんな聞いてくれ！　今迄みんなが激光電脳倶楽部だと思っていた物は実は激光倶楽部だったのだ！　激光倶楽部なんていうタイトルが本当だとするとただでさえ怪しいタイトルが更に怪しく、すてきで、すごくて、そほうで、やひで、かんそなイメージがありますね。思わせぶりな態度で一刀両断出来そうです。あ、これは紹介文なので紹介します。わたしもこのディスクマガジンのおかげであのききかいひ能力を手に入れる事ができそうです。きみは勉強ふそくでござる。にゃにいってんの。

推薦人は随時募集中でっす！

1999春号
ソフトバンクパブリッシング株式会社発行
定価 本体2,667円 +税

9784797309256

1929455026672

Printed in Japan クニメディア

ISBN4-7973-0925-3 C9455 ¥2667E

雑誌 65814-57